Pós-Graduação em Ciência da Computação

Antonio Braz Silva Finizola

DOGFOODING ANALYSIS SYSTEM: um sistema de análise de feedbacks de dogfooding para auxiliar as atividades de Testes Exploratórios

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2019

Antonio Braz Silva Finizola

DOGFOODING ANALYSIS SYSTEM: um sistema de análise de feedbacks de dogfooding para auxiliar as atividades de Testes Exploratórios

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

Área de Concentração: Inteligência computacional

Orientador (a): Profa. Dra. Flávia de Almeida Barros

Recife
2019

Catalogação na fonte
Bibliotecária Arabelly Ascoli CRB4-2068

F498d Finizola, Antonio Braz Silva
Dogfooding analysis system: um sistema de análise de feedbacks de dogfooding para auxiliar as atividades de testes exploratórios / Antonio Braz Silva Finizola. – 2019.
103.: il., fig., tab.

Orientadora: Flávia de Almeida Barros
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação. Recife, 2020.
Inclui referências.

1. Inteligência computacional. 2. Testes exploratórios. 3. Dogfooding. 4. Aprendizagem de máquina. I. Barros, Flávia de Almeida (orientadora). II. Título.

515 CDD (22. ed.) UFPE-CCEN 2020-34

Antonio Braz Silva Finizola

“Dogfooding Analysis System: um sistema de análise de feedbacks de dogfooding para auxiliar as atividades de Testes Exploratórios”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 30 de julho de 2019.

BANCA EXAMINADORA

_____

Prof. Dr. Alexandre Cabral Mota
Centro de Informática/UFPE

_____

Prof. Dr. Renato Fernandes Corrêa
Departamento de Ciência da Informação/UFPE

_____

Profa. Dra. Flávia de Almeida Barros
Centro de Informática/UFPE
(**Orientadora**)

AGRADECIMENTOS

Primeiramente, agradeço ao Senhor Jesus Cristo de Nazaré, meu Senhor e meu Salvador, por todas as bênçãos que Ele tem me proporcionado, sendo esta mais uma conquista que obtive por Sua Graça. A Ele seja a honra, a glória, a força e o poder para todo o sempre. Amém.

À minha mãe, Maria de Lourdes Finizola, e ao meu pai, Paulo Sérgio Finizola (in memória), pelo apoio e amor incondicionais, pelas orações, pelo aconchego e por todo o incentivo dado a mim para que eu pudesse alcançar objetivos que jamais imaginei que poderia conseguir. Eu amo vocês profundamente, meus pais. Que O Senhor os abençoe e os guarde poderosamente.

À minha noiva e futura esposa Mayara Araújo, pelo seu muito amor e carinho, além da toda compreensão a mim demonstrada nos momentos em que precisei ficar um pouco isolado para concluir meu trabalho. Obrigado por ser quem você é, meu bem. Te amo muito. Que O Senhor te abençoe.

Aos meus amigos e irmãos em Cristo Jesus, em especial a Mailton Fernandes, Renan Ferreira, Cloves Lima e Ivan Santos pelo o apoio e incentivo. Deus os abençoe.

À minha orientadora, professora Dra. Flávia de Almeida Barros, por toda sua atenção, direcionamentos e orientações, que me foram extremamente úteis para a conclusão deste trabalho. Seu papel foi imprescindível para a minha carreira. Muito obrigado por tudo, professora. Deus a abençoe.

Agradeço ao Projeto CIn/Motorola (um convênio entre o Centro de Informática da UFPE e a Motorola Mobility), que me proporcionou o ambiente e todos os meios necessários para que este trabalho fosse realizado. Deixo aqui meu muito obrigado!

RESUMO

Os Testes Exploratórios (TEs) configuram uma abordagem eficiente na área de Testes de Software, utilizada para a detecção de bugs inesperados ou desconhecidos. Os TEs são dinâmicos, uma vez que os testadores os projetam e os modificam à medida que os executam. Assim, é importante que os testadores disponham de fontes que forneçam informações úteis para melhor direcionar os TEs, de modo a garantir sua efetividade. Uma importante fonte de informação vem da adoção da abordagem *Dogfooding*, que é o uso intensivo dos produtos de software pelos colaboradores de uma empresa, antes de serem lançados no mercado. Algumas empresas que usam essa abordagem criam comunidades privadas para que os colaboradores postem seus feedbacks sobre os produtos, relatando críticas, elogios, sugestões e defeitos de software. Os testadores, então, podem utilizar as informações contidas nos feedbacks de defeitos a fim de melhorar a qualidade dos testes, isto é, detectar novos defeitos relacionados com base nos insights obtidos dos feedbacks. No entanto, nem sempre as empresas dispõem de métodos para identificar esses feedbacks úteis, sendo necessário análises manuais nas comunidades, que levam tempo e são pouco produtivas. Este trabalho, desenvolvido no contexto de uma colaboração entre o CIn-UFPE e a Motorola Mobility, teve por objetivo criar um processo para automatizar boa parte das atividades de análise de *feedbacks* de Dogfooding, a fim de obter informações úteis para direcionar as atividades de TEs. O protótipo do sistema implementado utiliza conceitos de Aprendizagem de Máquina (AM) e Recuperação de Informação (RI), contando com 4 módulos principais: (1) Obtenção dos feedbacks por aparelho; (2) Classificação de feedbacks (entre relevantes e irrelevantes); (3) Recuperação e seleção de charters (documentos que contêm as diretrizes de testes, que são selecionados a partir da análise dos feedbacks relevantes); (4) Criação e submissão da análise de dogfooding. O classificador foi construído usando AM, tendo sido testados diferentes algoritmos para este fim. Esse classificador é periodicamente atualizado (retreinado) de forma semiautomática, para não se tornar obsoleto (versão *beta*). Os testes realizados com o protótipo, principalmente os de classificação e usabilidade, apresentaram resultados muito positivos: foi possível obter um modelo de classificação com valores de acurácia e F1-score de 87% e 88% respectivamente; também foi possível obter um ganho de tempo de 47.5% e 64.29% em comparação com os procedimentos manuais nos dois testes de usabilidade realizados, além de um ganho de 55.56% de feedbacks relevantes identificados em um desses testes.

Palavras-chave: Testes Exploratórios. Dogfooding. Aprendizagem de Máquina. Recuperação de Informação.

ABSTRACT

The Exploratory Tests (ETs) configure an efficient approach in the Software Testing area, used for the unknown bugs detection. ETs are dynamic, as testers design and modify them as they execute them. Thus, it's important that testers have sources that provide useful information to better target ETs in order to ensure their effectiveness. An important information source comes from adopting the Dogfooding approach, which is the intensive use of software products by a company's employees before they are released. Some companies that use this approach create private communities for employees to post feedback on products, reporting criticism, praise, suggestions, and software defects. The testers can then use the information contained in the defect feedbacks to improve the quality of the tests, that is, to detect new defects related based in the feedback insights. However, companies don't always have methods to identify such useful feedbacks, and manual reviews in the communities are needed, that are time consuming and unproductive. This work, developed in the context of a collaboration between CIn-UFPE and Motorola Mobility, aimed to create a process to automate a good part of Dogfooding's feedback analysis activities, in order to obtain useful information to direct the ETs activities. The prototype of implemented system uses Machine Learning (ML) and Information Retrieval (IR) concepts, with 4 main modules: (1) Device feedbacks obtaining; (2) Feedbacks classification (between relevants and irrelevants); (3) Recovery and selection of charters (documents containing test guidelines, which are selected from the analysis of relevant feedbacks); (4) Creation and submission of dogfooding analysis. The classifier was constructed using Machine Learning, and different algorithms were tested. This classifier is periodically updated (retrained) semi automatically, so as not to become obsolete (*beta version*). The tests performed with the prototype, especially those of classification and usability, presented very positive results: it was possible to obtain a classification model with values of accuracy and F1-score of 87% and 88% respectively; it was also possible to obtain a time gain of 47.5% and 64.29% compared to the manual procedures in the two usability tests performed, in addition to a gain of 55.56% of relevant feedbacks identified in one of these tests.

Keywords: Exploratory Testing. Dogfooding. Machine Learning. Information Retrieval.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AM	Aprendizagem de Máquina
DF	Dogfooding
KDD	Knowledge Discovery in Databases
KDT	Knowledge Discovery from Texts
KNN	K Nearest Neighbors
MD	Mineração de Dados
MT	Mineração de Textos
NB	Naïve Bayes
RegEx	Regular Expressions
RI	Recuperação de Informação
SRIs	Sistemas de Recuperação de Informação
SVM	Support Vector Machines
SWEBOK	Software Engineering Body Of Knowledge
TE	Teste Exploratório
TEs	Testes Exploratórios
TF-IDF	Term Frequency – Inverse Document Frequency

SUMÁRIO

1 INTRODUÇÃO

Testes de Software configuram um dos principais processos da Engenharia de Software, cujos principais objetivos são a detecção e a correção de defeitos em produtos de software, de modo que os mesmos possam operar devidamente e com a qualidade esperada. Dentre as abordagens de testes, estão os Testes Exploratórios (TEs). De acordo com Bach (2003),

> Qualquer teste em que o testador controla ativamente o projeto de testes à medida que esses testes são realizados e usa as informações obtidas durante a execução deles para projetar testes novos e melhores (BACH, 2003, tradução nossa).

Essa abordagem tem como característica principal a não utilização de scripts prévios, pois são os testadores quem criam e executam seus próprios testes à medida que se familiarizam e exploram o produto de software. Outra característica importante é a capacidade de detectar defeitos precisamente e com mais rapidez, sendo que boa parte desses defeitos são detectados em cenários inesperados e/ou desconhecidos. Aqui, são os testadores quem gerenciam, de maneira ativa, a execução dos testes, sem a dependência de especificações prévias de como os testes devem ser executados.

O Teste Exploratório é uma atividade disciplinada, e depende de diversos critérios de planejamento como qualquer outra atividade intelectual de Engenharia de Software. Os testadores precisam, além de apropriarem-se dos produtos de software e considerar as diretrizes iniciais ao realizar seus testes, também precisam dispor de fontes de conhecimentos úteis para que sejam desenvolvidos novos insights e estratégias de testes diferenciadas, à altura dos cenários inesperados, os quais favorecem a detecção de bugs mais críticos.

Com base nisso, entra o conceito de *Dogfooding*. Essa abordagem configura uma importante fonte de conhecimento no campo de *Qualidade de Software*, pois contribui fortemente na captura de bugs críticos, uma vez que os produtos de software de uma dada empresa são utilizados intensivamente pelos colaboradores (como se fossem os usuários), antes de serem lançados no mercado. Os feedbacks fornecidos pelos colaboradores contêm informações que podem ser muito úteis na realização de testes exploratórios mais direcionados, levando em conta as situações anormais e de defeitos mencionadas nesses feedbacks.

1.1 TRABALHO REALIZADO

A motivação inicial deste trabalho de mestrado foi investigar como a abordagem de Dogfooding é utilizada no contexto de Testes de Software Exploratórios, com o objetivo de propor e desenvolver soluções computacionais para resolver possíveis problemas que fossem levantados. Para isso, foi utilizado o ambiente de um projeto de pesquisa em Testes de Software de uma empresa especializada em dispositivos móveis (Projeto CIn/Motorola), a qual possui uma política interna de dogfooding. A equipe de TEs do projeto realiza análises frequentes de feedbacks de dogfooding fornecidos pelos colaboradores, a fim de obter informações úteis que possam ser utilizadas como diretrizes de teste para detecção de novos defeitos.

Alguns problemas nesse processo de obtenção de informações foram levantados. Por exemplo, as comunidades não possuem mecanismos que ajudem os testadores a identificarem se feedbacks são úteis (i.e., que mencionam defeitos) ou não, e precisam gastar um tempo razoável para busca-los e mapeá-los. Outro caso é que alguns dos feedbacks úteis não são contemplados nas buscas, devido à dificuldade e o tempo gasto para identificar todos eles ou boa parte deles. Os testadores ainda precisam selecionar os *charters* (i.e., *missões*, documentos com informações a respeito do que deve ser considerado durante a realização dos testes, ou seja, as *diretrizes de teste*) relacionados às áreas apontadas nos feedbacks e reportar as análises realizadas. Esses procedimentos são manuais, tornando as atividades lentas e e com pouca qualidade.

Diante disso, foi proposto um processo para automatizar boa parte desses procedimentos manuais. Esse processo consiste em classificar automaticamente os feedbacks, de modo a fornecer os que são relevantes aos testadores diretamente; mapear automaticamente os possíveis charters relacionados aos feedbacks, para seleção; e permitir a submissão semiautomática da análise. Dessa forma, esse processo foi implementado num protótipo de um sistema computacional, que utiliza conceitos de *Aprendizagem de Máquina* (AM) e *Recuperação de Informação* (RI). O objetivo principal desta iniciativa foi tentar proporcionar um ganho significativo na redução do tempo gasto nas análises, principalmente na identificação de feedbacks relevantes que não são mapeados manualmente.

Nesta pesquisa, foram comparados 3 algoritmos de AM, sobre uma base com 2691 amostras (particionada entre *treino* e *teste*) de feedbacks relevantes e

irrelevantes, utilizando os procedimentos de Classificação Textual, Seleção de Atributos e Otimização de Parâmetros. Testes preliminares com a base de *treino* foram realizados para comparar os modelos, seguidos pelos testes de validação com a base de *teste*. Testes estatísticos adicionais foram realizados para avaliar a qualidade dos modelos. O objetivo foi obter um modelo geral que fosse capaz de classificar bem novas amostras de feedbacks, de modo a substituir a classificação manual realizada pelos testadores.

Também foram realizados testes no processo de seleção de charters para os feedbacks, para verificar a correta recuperação dos charters, isto é, se os charters recuperados estão de fato relacionados aos feedbacks.

Ao final, foram realizados os testes de usabilidade, que consistiram na realização de análises de dogfooding com e sem o uso do protótipo. O objetivo foi verificar o ganho de tempo e a capacidade de identificação dos feedbacks relevantes em comparação com os procedimentos manuais.

Os resultados demonstraram que a iniciativa é válida, e que contribuiu para a redução do tempo gasto nas análises e também na detecção de mais feedbacks úteis, melhorando os procedimentos de análises de dogfooding.

1.2. ESTRUTURA DO DOCUMENTO

O presente trabalho está dividido em 5 capítulos, com exceção da Introdução. Eles estão organizados da seguinte forma:

- Capítulo 2 – apresenta uma breve bibliografia sobre os fundamentos de Testes Exploratórios: seus conceitos básicos, sua utilização e benefícios. Além disso, também é apresentada a abordagem *Dogfooding*;
- Capítulo 3 – apresenta brevemente os fundamentos da Mineração de Textos e suas fases. Também é apresentada uma seção sobre *classificação de textos*, incluindo os algoritmos utilizados neste trabalho;
- Capítulo 4 – apresenta o problema identificado, bem como a solução proposta. Mais precisamente, é apresentado o processo geral do protótipo desenvolvido e a descrição dos seus módulos principais;

- Capítulo 5 – apresenta a implementação de cada funcionalidade do protótipo em detalhes. Além disso, também apresenta todos os testes realizados: classificação, testes estatísticos, validação e testes de usabilidade.
- Capítulo 6 – apresenta as contribuições deste trabalho e as propostas de trabalhos futuros.

2 TESTES EXPLORATÓRIOS

Segundo Itkonen e Rautiainen (2005), desenvolvedores e testadores de software sempre realizaram Testes Exploratórios (TEs). A abordagem de TEs difere essencialmente das demais abordagens de testes, pois não necessitam de detalhes ou especificações prévias para que possam ser executados, como é o caso de scripts de testes (i.e., testes guiados por um passo a passo prévio, que possui um conjunto de especificações que informam ao testador como o teste deve ser executado) utilizados nessas abordagens. Eles também são caracterizados por sua dinamicidade, que visa projetar, testar e modificar o design de teste como parte da sua própria execução, ao invés de ter uma fase específica para cada um desses procedimentos antes da execução. Por exemplo, um testador pode projetar vagamente um teste e, à medida que a aplicação ou o software vão sendo “explorados” e testados, o testador pode reformular o teste para atender os novos cenários identificados com base na exploração inicial e testar nova e sucessivamente.

O objetivo dos TEs não é de substituir as demais abordagens de testes, mas complementá-las (CRAIG; JASKIEL, 2002). Esses complementos se referem principalmente às situações adversas (cenários alternativos de teste, não mapeados, mas que são capazes de detectar bugs precisamente) que surgem ao longo dos procedimentos de teste. Em outras palavras, “é uma etapa crítica na determinação da qualidade geral de uma aplicação porque serve para detectar bugs inesperados ou desconhecidos.” (DOTTERWEICH, 2017, tradução nossa).

Há também a abordagem de *Dogfooding*, que configura uma importante fonte de conhecimento no campo de *Qualidade de Software*, e contribui fortemente na identificação de bugs críticos, uma vez que os produtos de software de uma dada empresa são utilizados intensivamente pelos colaboradores (como se fossem os usuários), antes de serem lançados no mercado. Portanto, essa abordagem tem forte ligação com os objetivos dos Testes Exploratórios.

As seções a seguir tratam desde os conceitos básicos sobre TEs; os cenários de aplicabilidade para TEs; e os benefícios proporcionados. Também foi colocada uma seção sobre *Dogfooding*, com explicações mais detalhadas de como essa abordagem é utilizada.

2.1 CONCEITOS BÁSICOS

Teste Exploratório é uma abordagem da área de Testes de Software, sendo definida como uma modalidade que amplia a autonomia, a liberdade e a responsabilidade do testador individual com o objetivo de aperfeiçoar gradativamente a qualidade das suas atividades (KANER et al., 2002). Os fundamentos dessa abordagem, segundo Bach (2003), consistem em: aprendizagem simultânea, projeto de testes e execução de testes. Esses fundamentos caracterizam-se como tarefas de suporte que são executadas em paralelo ao longo de um projeto de software.

Kaner et al. (1999) apresentam os TEs como um método de execução contínua de testes, isto é, manter um software em teste após a execução dos testes com script. Essa perspectiva tem o objetivo de diminuir os esforços e os investimentos necessários para a criação de testes com script (projeto de testes e documentação), considerando a instabilidade do software em questão, o qual passa por modificações constantes.

Kaner et al. (2002) relacionam as atividades de TEs como sendo uma missão geral sem, no entanto, dispor de um caminho pré-definido para realizá-la. Essa missão é denominada de *charter*. Um *charter* é um artefato que indica as diretrizes iniciais que devem ser consideradas antes de iniciar os testes exploratórios. Ele pode definir quais documentos estão disponíveis para auxílio, quais táticas devem ser utilizadas, e quais riscos estão relacionados. O charter não define o passo a passo de como testar, mas restringe a atenção do testador e fornece direcionamentos que o ajuda a melhor testar o software em questão (COPELAND, 2004) (TINKHAM; KANER, 2003).

Kaner et al. (2002) mencionam ainda que o TE envolve o aprendizado e a realização de diversos experimentos simultaneamente. O aprendizado vem do fato de o testador ser capaz de desenvolver suas habilidades cognitivas, que o impulsionam a mapear diversos fluxos improvisados de teste. Os experimentos configuram cada um dos caminhos que o testador tomou para a localização de defeitos, e a avaliação de quais caminhos renderam melhores resultados. Assim, vemos que o testador conseguirá criar testes muito mais robustos e sofisticados dos que ele criou inicialmente em função dos conhecimentos obtidos continuamente e dos experimentos realizados durante as execuções.

Segundo Craig e Jaskiel (2002), os resultados de um teste ou de uma campanha de testes são alguns dos artefatos que conduzem o testador, na maioria

das vezes, a aprofundar-se (i. e., explorar) numa área específica do software em teste. Também afirmam que a beleza do TE é que as áreas mais produtivas de teste se expandem com grande rapidez.

Bach (2003) menciona que praticamente todos os testes realizados pelos testadores possuem um certo *grau de exploração*. Existe uma *continuidade* entre os testes essencialmente roteirizados (os que são executados obedecendo o passo a passo de um script) e os testes exploratórios (onde os *insights* surgem à medida que os testes vão sendo executados). Qualquer esforço de teste se enquadra em algum lugar dentro desses aspectos. O foco desse esforço, portanto, revela a questão: até que ponto seus testes são exploratórios? Mesmo que se esteja lidando com testes com script puramente, um bom e habilidoso testador pode transformá-los em testes exploratórios, visto que ele permanece ativo quanto às indicações de erros presentes em um software. A presença ou a ausência dessas indicações de erros irá definir se o testador deverá continuar com os scripts ou parar para realizar testes exploratórios (TINKHAM; KANER, 2003).

Vejamos um exemplo do emprego dos TEs: para saber se um defeito foi consertado, suponha que o testador recorra às informações contidas nos relatórios de erros corrigidos, e execute uma campanha de *testes de regressão*[1] baseado neles. Durante a execução, um testador poderá realizar diversos testes adicionais, a fim de ter certeza de que os defeitos mapeados nos relatórios foram devidamente corrigidos. Percebe-se, no entanto, que não existem registros que detalham os fluxos alternativos que geraram os testes adicionais, pois todos eles foram criados arbitrariamente pelo testador durante suas atividades. Esses fluxos são os diversos caminhos percorridos pelo testador, que vão além dos caminhos indicados nos testes de regressão. Em outras palavras, cada fluxo alternativo configura um novo teste adicional ou um caminho que conduz à realização desse novo teste adicional. Dessa forma, o testador está realizado uma “exploração”, isto é, ele está executando um *teste exploratório*.

Bach (2000) também salienta que a abordagem de TE é voltada sobretudo à localização de erros, ou seja, encontrar defeitos e documentar os resultados dos testes é mais importante do que planejar e criar antecipadamente scripts dos caminhos de execução de testes. Vale ressaltar, porém, que os TEs são tarefas

[1] Técnica de teste em que são aplicadas as novas versões do software, com objetivo de verificar se as correções feitas nessas versões não trouxeram novos defeitos. Configura uma abordagem de teste baseada em scripts.

disciplinadas (BACH, 2004) e apesar de enfatizar a autonomia dos engenheiros de teste, um determinado grau de planejamento é necessário. Esse planejamento pode compreender, por exemplo: análise de recorrência de defeitos, com base nos que foram recentemente reportados; análise das correções integradas na versão corrente do software; identificação dos charters apropriados, isto é, os charters que contêm as diretrizes que melhor atendem ao cenário corrente do que precisa ser testado, com base nas análises realizadas. Esses critérios de planejamento apresentados sugerem uma alternativa para a execução de TEs melhor direcionados, com base nos principais problemas levantados até o momento.

Os TEs também se caracterizam mais como um modo de pensar sobre o teste do que uma metodologia (ABREU et al. 2016, apud KANER et al., 2009). Reuniões e *brainstorming* podem contribuir positivamente na produção de melhores ideias e também para desenvolver bons e novos *insights*. Um dos principais métodos de execução de testes exploratórios consiste em descrever o que se vê, reconhecer e gerenciar as tendências e utilizar ferramentas e mecanismos (checklist, relatórios de erros, release notes, feedbacks de usuários, etc.) para facilitar as atividades, contribuindo assim na geração de situações interessantes para explorar.

Segundo SWEBOK (IEEE, 2014):

> A eficácia do teste exploratório depende do conhecimento do engenheiro de software, que pode ser derivado de várias fontes: comportamento do produto observado durante o teste, familiaridade com o aplicativo, a plataforma, o processo de falha, os tipos de possíveis faltas e falhas, o risco associado a um produto específico e assim por diante (IEEE, 2014, tradução nossa).

Tinkham e Kaner (2003) também pontuam que a abordagem de TEs é fortemente baseada em conhecimento. Os testadores podem utilizar diversas fontes de conhecimento para aprimorar e aumentar a qualidade dos seus testes. Alguns testadores apuram experiências nos casos em que, num determinado projeto, conseguiram localizar bugs a partir de cenários específicos, e tentam levar esses cenários para testes futuros em outros projetos. Há também testadores que se baseiam nas experiências de outros testadores, não havendo a necessidade de experimentarem a ocorrência dos bugs em si. As falhas identificadas em uma dada aplicação, cujas as informações estão disponíveis em fontes publicadas, também configuram uma fonte de informação aos testadores.

Com base em todas essas afirmações sobre TEs, Itkonen e Rautiainen (2005) resumem as 5 propriedades que os descrevem: (1) não é definido com antecedência e também não utiliza scripts. Caracteriza-se como uma exploração a partir de uma missão geral, porém sem instruções específicas de como realizá-la; (2) é guiado pelos resultados de testes anteriores e pelo conhecimento obtido a partir deles. Qualquer informação disponível para atingir a meta de um teste é útil para um testador de TE; (3) o principal objetivo é encontrar defeitos por exploração, ao invés de criar um conjunto de casos de teste com scripts para uso posterior; (4) o TE é a aprendizagem simultânea do software em teste, do projeto de teste e da execução do teste; e a (5) a eficácia do teste depende do conhecimento, das habilidades e da experiência do testador.

2.2 APLICABILIDADE DE TEs

Abreu et al. (2016) afirmam que TEs são especialmente úteis em situações onde existem testes complexos, pouco conhecimento sobre o produto/software ou em casos onde eles são parte do planejamento de um conjunto de testes com script. Também colocam como regra básica que a qualquer momento o TE pode ser utilizado, ou quando o próximo teste a ser executado possui um cenário simplório ou quando se quer pensar “fora da caixa”. Salientam também a necessidade da prática, uma vez que ela amplia a perspicácia do testador e contribui para o desenvolvimento de habilidades estratégicas através da exploração e da experiência.

Bach (2001) apresenta alguns argumentos que explicam em quais momentos os TEs se encaixam. “O teste exploratório é tudo o que você tem no começo...” (BACH, 2001). O autor enfatiza que os TEs são a única alternativa no início de qualquer projeto de teste, uma vez que os testes (i.e., as abordagens com scripts) ainda não foram criados/implementados para a tecnologia que está sendo desenvolvida. Também afirma que, mesmo que os testes já existam (i.e., *testes de regressão* e demais abordagens que utilizam scripts), é necessário que o testador se familiarize com o produto e que os scripts de cada teste sofram constantes revisões e atualizações. Escrever/atualizar testes também configura um procedimento *exploratório*.

Uma vez que uma variedade de testes é produzida em função dos procedimentos iniciais, ainda há espaço para os TEs. O autor incentiva que, mesmo que os testadores possuam muitos testes com script para realizar, eles podem utilizar

técnicas improvisadas e cobrir diferentes partes do produto/software a partir dos insights iniciais presentes nos próprios testes com script. A ideia consiste em trabalhar as muitas variações dos testes, e a seguir, dar continuidade aos procedimentos com script.

O autor propõe que durante todo o projeto haja questionamentos por parte dos testadores em relação aos testes a eles atribuídos, visto que nenhum processo de teste oferece uma cobertura de 100%. Também propõe alocar um tempo razoável em cada ciclo de teste para a execução de testes exploratórios puros, de modo a aumentar a abrangência e a profundidade dos testes. O autor também propõe que haja um testador dedicado ou testes específicos para tarefas contínuas, para a execução de testes exploratórios puros.

Resumindo, qualquer cenário no qual não se saiba qual deve ser o próximo teste, ou quando se quer ir além dos testes óbvios, é um cenário que satisfaz a aplicação de TEs (BACH, 2003). Dessa forma, os TEs se encaixam principalmente quando se quer: (1) fornecer feedback rápido sobre um novo produto ou recurso; (2) diversificar casos de testes; (3) localizar bugs mais importantes com o menor espaço de tempo possível; (4) monitorar as atividades de outros testadores, realizando investigações breves; (5) investigar e isolar um defeito em particular; e (6) monitorar o status de um risco particular, a fim de avaliar a necessidade de execução de testes com scripts nessa área.

Copeland (2004) afirma que os TEs também são aplicáveis em qualquer situação em que o próximo caso de teste deve ser baseado em testes e resultados anteriores. Também é eficiente na exploração das dimensões, do escopo e das variações relacionados a um defeito encontrado, de modo a fornecer um melhor feedback aos desenvolvedores.

Bach (2003) menciona ainda que a eficiência do TE também se deve ao fato de como as informações fluem da execução dos testes para o replanejamento dos testes. Quando esse fluxo de informações é fraco ou lento, essa eficiência tende a diminuir, devendo-se recorrer aos testes com script. Mesmo nesses casos, é importante considerar sempre o uso de estratégias exploratórias combinadas às estratégias com script.

2.3 BENEFÍCIOS DOS TEs

Itkonen e Rautiainen (2005) afirmam que o TE tem sido considerado uma abordagem muito útil em Testes de Software diante dos vários benefícios que ele pode fornecer. Sommerville (2005, apud ABREU et al., 2016) afirma que:

> Um teste bem-sucedido para detecção de defeitos é aquele que faz com que o sistema opere incorretamente e, como consequência, expõe um defeito existente, enfatizando assim um fato importante sobre os testes (SOMMERVILLE, 2005, apud ABREU et al., 2016).

Craig e Jaskiel (2002) afirmam que, em testes com script, um testador cria muitos testes que aparentemente pareciam ser úteis durante o design, mas não o são durante suas execuções, devido eles não revelarem defeitos. Bach (2003) afirma também que:

> O que torna o teste exploratório interessante [...] é que quando um testador tem as habilidades para ouvir, ler, pensar e relatar, rigorosa e efetivamente, sem o uso de instruções pré-scriptadas, a abordagem exploratória de teste pode ser muitas vezes tão produtiva (em termos de revelar informações vitais) quanto a variedade programada. [...] até os testadores sem habilidades especiais podem produzir resultados úteis que não seriam previstos por um script (BACH, 2003, tradução nossa).

Vale lembrar que os TEs não são uma alternativa para a substituição das técnicas com script, como diz o próprio Bach (2003). O autor menciona que, em alguns contextos, uma abordagem a nível de script alcançará melhores resultados; em outros casos, os melhores resultados serão obtidos com uma abordagem exploratória, isto é, criação e otimização de testes à medida que são executados. O autor também diz que na maioria das situações, os maiores benefícios são obtidos través de uma mistura de ambas abordagens.

Agora, considerando os pontos levantados por Itkonen e Rautiainen (2005), Naseer e Zulfiqar (2010), Itkonen (2011) e Petroski (2009), é possível fazer resumo dos benefícios provenientes da adoção da abordagem de TEs. A seguir, citaremos os principais deles.

O primeiro benefício, sendo considerado o mais elementar dos TEs, é a capacidade de aumentar a eficácia do teste em termos de número e importância de defeitos encontrados. Os TEs configuram um método eficaz para identificar casos particulares de teste, os quais não são fáceis de serem percebidos com a utilização

das demais abordagens. Em outras palavras, TE contribui fortemente na detecção de erros críticos com maior agilidade e precisão do que qualquer outra abordagem.

O segundo benefício é a investigação ativa em relação aos riscos que circundam um produto. Durante a familiarização com o produto em questão (i.e., como as falhas podem ser geradas, quais são os usuários finais, qual o impacto de um bug A ou B, etc.), o testador explorador busca continuamente mais e mais informações sobre o produto e seus riscos, e inicia o projeto de testes exploratórios com base no conhecimento que adquiriu até o momento com essas análises contínuas. Com isso, ele poderá utilizar quaisquer técnicas de teste, de modo a continuar aprendendo mais precisamente sobre o produto e suas falhas, criando cenários de testes melhores e mais poderosos à medida que aprende.

Um terceiro benefício é minimizar a documentação de um produto/software antes de executar os testes. Essa vantagem diz respeito às situações em que os requisitos e o design do produto/software mudam com muita frequência, ou nos casos onde o sistema está no estágio inicial de desenvolvimento (i.e., algumas partes do sistema são implementadas, mas a probabilidade de grandes mudanças ainda é alta).

Um quarto benefício citado é a possibilidade de realizar testes sem a necessidade a priori de requisitos abrangentes ou documentação de especificação, uma vez que os testadores de TEs podem facilmente utilizar toda a experiência e o conhecimento obtido de várias outras fontes de informação. Isso contribui para economizar muito tempo e esforço dos testadores durante a atividade de teste.

Um quinto benefício é o rápido fluxo de feedback dos testes entre desenvolvedores e testadores. Esse *loop* de feedback é especialmente rápido, visto que os testadores exploratórios podem reagir rapidamente às alterações no produto/software e fornecer aos desenvolvedores resultados de teste com maior agilidade.

2.5 DOGFOODING

Dogfooding (DF) é uma abordagem compartilhada entre empresas de desenvolvimento de software que testam seus próprios produtos ou serviços, de modo que seus colaboradores os utilizam em seu dia a dia (HARRISON, 2006; LAFELDT, 2017). O principal objetivo é que, por meio do uso do seu próprio software/produto, as empresas possam validá-lo e melhorá-lo antes de dispô-lo aos usuários finais. Assim, essa prática funciona como um mecanismo de garantia de *controle de qualidade*. DF também contribui fortemente nas questões de marketing, uma vez que uma empresa demonstra confiança no que ela produz (LAFELDT, 2017; HARRISON, 2006).

Segundo Evanhaim (2016, tradução nossa), o "*dogfooding* tornou-se uma prática efetiva em empresas de software proeminentes, para testes e feedback eficazes". No contexto de aplicações para dispositivos móveis, o autor menciona que essa abordagem valoriza bastante o desenvolvimento de software, principalmente pelo fato de submeter essas aplicações a um uso intensivo, de modo a tornar as aplicações mais estáveis. O autor também afirma que quando um usuário final encontra um problema numa aplicação qualquer, não lhe resta muita coisa a fazer além de fornecer um feedback negativo a respeito da aplicação e possivelmente desinstalá-la.

No caso dos *dogfooders*, que nesse contexto seriam desenvolvedores, testadores e demais colaboradores, qualquer aspecto duvidoso pertinente à aplicação seria devidamente investigado, contribuindo assim para que as aplicações se tornem isentas de problemas vagos (EVANHAIM, 2016). Ainda considerando o contexto de aplicações móveis, Abul-Ezz (2018) menciona que a prática de *dogfooding* tem como desafio solucionar problemas que envolvem testadores capazes de fornecer feedback de qualidade, de modo a minimizar a distância entre testadores e usuários dessas aplicações.

No trabalho de Rahman et al. (2015), são pontuadas algumas das práticas mais utilizadas no *desenvolvimento contínuo de software*. Uma delas é o *staging*, que os autores definem como sendo um conjunto de técnicas adotadas em duas situações: (1) implementação e testes das mudanças feitas no software; e (2) a integração das mudanças para os usuários finais. O *dogfooding* entra como uma das técnicas utilizadas em *staging*. Como exemplo, os autores citam um modelo de *dogfooding* que consiste na utilização de servidores de produção acessíveis apenas pela equipe de

software para implantação das mudanças, em que alguns membros são responsáveis por testar essas mudanças como se fossem usuários finais.

Defranco (2014) e Harrison (2006) apontam também que o benefício mais citado dessa abordagem diz respeito à capacidade de capturar bugs, uma vez que o uso frequente do produto conduz a caminhos além dos limites propostos por checklists de *Quality Assurance*. Assim, o uso frequente do produto garante a verificação de todos os seus aspectos, contribuindo para o aumento das chances de deparar-se com os defeitos imprevisíveis (DEFRANCO, 2014). Os autores também afirmam que os feedbacks internos sempre devem ser levados em conta, pois constituem um artefato que contribui fortemente para o desenvolvimento de produtos com qualidade.

Mäntylä e Itkonen (2014) apresentam, em seu survey, como se dão as atividades de detecção de defeitos de software no nível organizacional. O foco dos autores é a abordagem denominada *detecção implícita de defeitos*, que, "[...] avalia a qualidade do produto e detecta defeitos enquanto se trabalha em direção a algum outro objetivo principal." (MÄNTYLÄ; ITKONEN, 2014, tradução nossa). Os autores afirmam que as atividades de *dogfooding* estão diretamente relacionadas à *detecção implícita de defeitos*. Também afirmam que *dogfooding* é, provavelmente, uma das atividades mais adotadas e amplamente difundidas em relação à *detecção implícita de defeitos*. Além disso, esse trabalho apresenta algumas questões em relação aos *documentos utilizados em detecção de defeitos*. Na Tabela 1, é possível ver uma lista dos documentos utilizados nesse contexto.

Documento	Nº de indivíduos	Total de defeitos	TDD/TDR[2] (%)	Média de defeitos
casos de teste	58	804	18.2	13.9
notas de versão	31	206	4.7	6.7
manual do produto	54	507	11.4	9.4
apresentação do produto e materiais de treinamento	29	138	3.1	4.7
requisitos e especificações de características	66	711	16.1	10.8
especificação técnica do	42	351	7.9	8.4

[2] TDD/TDR: todos os defeitos por documento dividido por todos os defeitos relatados.

produto				
mensagens ou relatórios[3]	61	969	21.9	15.9
sem uso de documentos	37	685	15.5	18.5
outros	2	56	1.3	27.8
total	380	4427	100	11.7

Tabela 1: Detecção de defeitos através do uso de documentos
Fonte: Adaptado de Mäntylä e Itkonen (2014)

Esse é o resultado das investigações feitas pelos autores sobre o papel que a documentação desempenha na detecção de defeitos. Eles ressaltam ainda que a indicação de "documento mais utilizado" depende de pontos de vista: número de pessoas usando o documento, número total de defeitos detectados com uso do documento ou o nº médio de defeitos detectados por pessoa usando o documento. Pela Tabela 1, os autores concluem que: (1) "especificações" são usadas pela maioria dos indivíduos (66); (2) "mensagens e relatórios de defeitos"[4] tem maior número de defeitos (969); "sem uso de documento" tem maior média (18.6) em termos de TDD/TDR. Os autores afirmam ainda que esses resultados desafiam o papel dominante dos *casos de teste* na detecção de defeitos de software, e pontuam várias fontes e variedades de conhecimento documentado.

Façamos agora uma relação entre *dogfooding* e TEs no contexto deste trabalho, com base nos pontos anteriores. Pensando nos feedbacks de *dogfooding* como sendo documentos, eles se encaixariam na categoria *mensagens ou relatórios*. Isso quer dizer que, se os documentos dessa categoria contribuem fortemente na detecção de bugs, os feedbacks de *dogfooding* também o fazem. Considerando então a natureza da abordagem de *dogfooding* (*detecção implícita de defeitos*), os bugs detectados em situações não previstas em certas áreas do software fornecem um forte indício de que há outros cenários a serem investigados. Em outras palavras, há uma probabilidade grande de que haja outros bugs ocultos nessas áreas. Dessa forma, as informações contidas nos feedbacks de *dogfooding* podem servir como *diretrizes* extras, de modo a ampliar o alcance dos testes na detecção de novos defeitos. O

[3] Compreende e-mails, relatórios de defeitos, requisição de usuários, etc.
[4] O número de defeitos detectados em *mensagens ou relatórios* deve-se, parcialmente, pela ampla gama de documentos que podem ser interpretados e/ou incluídos neste grupo.

objetivo seria, então, detectar *defeitos relativos* aos defeitos mencionados nos feedbacks.

Como exemplo, tomemos um exemplo de um feedback de *dogfooding* de uma comunidade interna de uma organização:

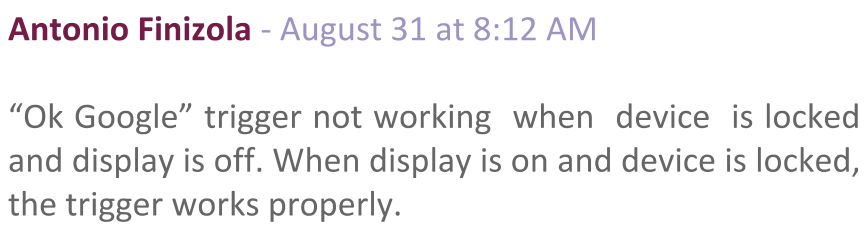

Figura 1: Exemplo de um feedback de dogfooding
Fonte: Autor (2018)

Neste exemplo, vemos que o comando de voz “Ok Google” não é acionado quando o display do aparelho está desligado. O defeito inicial está relacionado ao comando de voz, uma vez que deveria ser acionado independentemente de o display estar ou não ligado. A diretriz inicial indica que tanto os comandos de voz quanto o display devem ser investigados/explorados, abrindo espaço para a criação de novas diretrizes relacionadas. Por exemplo: testar outros comandos de voz; testar aplicações que funcionam em background (display desligado); testar comportamento do display em chamadas e notificações. A partir dessas diretrizes, os testadores podem compor novos cenários de testes que circundam o defeito inicial. Alguns exemplos de testes seriam: testar comando “Ligar para João” com display ligado e desligado; verificar comportamento do display com aplicação de música, testando alguns comandos de voz para troca, pausa de faixas; testar se numa chamada o display é acionado imediatamente; verificar comportamento do display quando o aparelho recebe uma notificação (i.e., mensagem de texto, e-mails). Assim, essa estratégia pode possibilitar a detecção de outros defeitos dentro de um mesmo contexto, a partir das diretrizes e dos cenários de testes identificados em função das informações iniciais obtidas com o feedback de dogfooding.

2.6 CONSIDERAÇÕES FINAIS

Os TEs são cruciais para a garantia de qualidade de software, principalmente no tocante à detecção de erros críticos, além de permitir a realização de investigações apuradas sobre os riscos inerentes ao produto de software. O conhecimento dos testadores é um dos parâmetros mais importantes que determina a eficiência dos TEs, e esse conhecimento pode ser obtido em diversas fontes (relatórios de erros, cenários de testes passados, apropriação do produto de software, etc.).

Também é importante considerar a intercalação das políticas de teste, ou seja, as iniciativas de aplicação de *testes com script* e *testes exploratórios*, contribuindo para uma cobertura de testes mais completa do produto de software, preservando ainda mais sua qualidade.

Outro ponto importante é o fato de os TEs serem atividades totalmente gerenciáveis, possuindo objetivos pontuais e procedimentos de controle e de planejamento. Mesmo nas abordagens que não obedecem um planejamento a priori (estilo livre), existem políticas de delegação das tarefas, bem como as diretrizes de como essas tarefas devem ser realizadas.

Também vale ressaltar que abordagem de *dogfooding* pode contribuir fortemente para a garantia de qualidade de um software, bem como para auxiliar no planejamento e na execução de testes exploratórios mais direcionados.

3 MINERAÇÃO DE TEXTOS

Em vários contextos organizacionais e empresariais, *análise de dados* configura uma das tarefas mais cruciais para o auxílio à tomada de decisão. Segundo Fayyad et al. (1996), o *método convencional* para transformação dos dados em informação consiste em tarefas manuais, onde esses dados são analisados e "processados" por especialistas de domínio, cujas informações geradas devem ser documentadas em relatórios para análises posteriores. Na grande maioria das situações, devido ao grande volume de dados a ser processado, esse método torna-se impraticável.

Nesse cenário, vários estudos foram desenvolvidos com o intuito de automatizar a extração de informações em grandes bases de dados. A partir daí, entra o conceito de *KDD* (*Knowledge Discovery in Databases* ou *Descoberta de Conhecimento em Bases de Dados*). A *KDD* caracteriza-se como um processo não trivial de identificação de *padrões* em dados, os quais devem ser válidos, novos e potencialmente úteis, com o objetivo de aprimorar a compreensão de um determinado problema ou de um procedimento de tomada de decisão (FAYYAD et al., 1996). A KDD está focada sobretudo em extrair informações importantes, implícitas em *dados estruturados*, de modo a obter conhecimentos úteis. No entanto, é necessário considerar o contexto de *dados não estruturados*.

O avanço das mídias digitais proporcionou um cenário oportuno para o armazenamento em massa de *dados não estruturados*. Grande parte desses é representada em formato textual, por exemplo: e-mails, textos livres obtidos por resultados de pesquisas, arquivos eletrônicos gerados por editores de textos, páginas da Web, campos textuais em bancos de dados, documentos eletrônicos em geral, etc. (ARANHA; PASSOS, 2006). O *texto* é um dos meios mais comuns de armazenamento de informação (TAN, 1999). Estima-se que 80% das informações de diversas organizações e de conteúdo online estejam em formato textual (TAN, 1999) (CHEN, 2001). Nesse contexto, entra em foco a área de *Mineração de Textos*, cujo objetivo é obter, automaticamente, informações úteis em grandes quantidades de dados textuais.

Este capítulo pretende apresentar os principais conceitos relacionados à *Mineração de Textos*, com foco na *classificação de textos*. O objetivo é apresentar os métodos e as técnicas utilizados para lidar com *dados não estruturados*.

3.1 DEFINIÇÕES

A *Mineração de Textos* (MT) ou *Descoberta de Conhecimento em Textos* (do inglês, *Knowledge Discovery from Texts* - KDT) surge da necessidade de se extrair automaticamente informações relevantes e conhecimentos úteis em *textos* (BARION; LAGO, 2008). Conhecimentos úteis são aqueles que podem ser aplicados de modo a apoiar algum processo de tomada de decisões, trazendo algum benefício (SOARES, 2008).

Como foi dito antes, a grande quantidade de informações disponibilizadas em linguagem natural faz da MT uma ferramenta poderosa, principalmente em termos de *gestão do conhecimento* (EBECKEN et al., 2003) (TAN, 1999). O objetivo principal da MT consiste na extração de características e na descoberta de padrões úteis a partir de uma grande quantidade de dados não estruturados (BRITO, 2016). Outras definições para Mineração de Textos, são:

> - MT configura um processo de extração de padrões úteis e não-triviais a partir de documentos textuais (TAN, 1999).
> - MT é um conjunto de técnicas e métodos usados na extração de conhecimentos a partir de dados não estruturados (TICOM, 2007).
> - MT é um método especializado de Mineração de Dados, onde a principal diferença está em lidar com dados não-estruturados (REZENDE et al., 2003).

Vejamos algumas definições de Mineração de Dados (MD) e as suas semelhanças com as de MT:

> - MD é a extração de padrões úteis não conhecidos a priori, a partir de grandes bases de dados (CASTANHEIRA, 2008).
> - MD é o processo não trivial de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis embutidos nos dados (FAYYAD, 1996).
> - MD é a exploração e a análise, por meio automático ou semiautomático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos (BERRY; LINOFF, 1997).

Com base nessas definições e nas anteriores, é possível perceber as semelhanças entre as duas áreas: ambas objetivam detectar e extrair padrões úteis de grandes quantidades de dados; compartilham das mesmas técnicas de mineração (e.g., classificação, agrupamento) (WEISS et al., 2010); e utilizam amostras de dados passados (MEDEIROS, 2004) com o objetivo de realizar inferências sobre novos

dados a partir dessas amostras. A principal diferença, no entanto, está no tipo de dado com o qual cada abordagem lida (REZENDE et al., 2003).

No Quadro 1 é possível ver um resumo das comparações entre as áreas de MD e MT.

	Mineração de Dados	**Mineração de Textos**
Tipo de dado	Dados categóricos e numéricos	Textos
Fonte dos dados	Bases de dados relacionais	Textos em formato livre
Objetivo	Prever resultados de situações futuras	Recuperar características representativas e categorizar resultados
Métodos	Classificação, agrupamento, redes neurais, regras, etc. Estão relacionados à MT: ontologias, linguística, redes neurais especiais, indexação, etc.	

Quadro 1: Mineração de Dados vs. Mineração de Textos
Fonte: Adaptado de Medeiros (2004)

Medeiros (2004) afirma que o processo de MT possui duas fases mais gerais: *extração de informação* e *mineração de dados* (ver Figura 2). Na primeira fase são realizados os tratamentos para representação dos dados. O objetivo é extrair conceitos, estatísticas e palavras importantes de um conjunto textual, de modo a obter uma estrutura mínima. Na segunda fase são aplicados os processos de mineração de dados (geração de regras, classificação, agrupamento etc.).

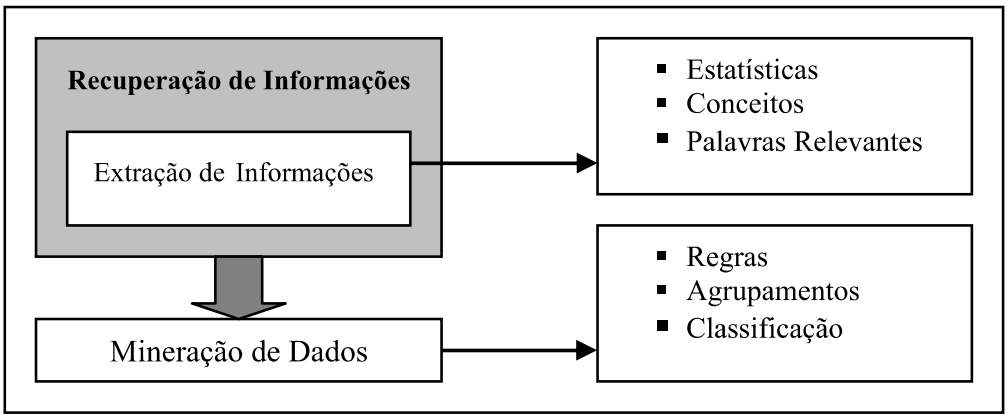

Figura 2: Processo da Mineração de Textos
Fonte: Medeiros (2004)

3.2 FASES DA MINERAÇÃO DE TEXTOS

Aranha (2007) apresenta uma metodologia para Mineração de Textos, constituída em 5 etapas: Coleta, Pré-processamento, Indexação, Mineração e Análise (Figura 3).

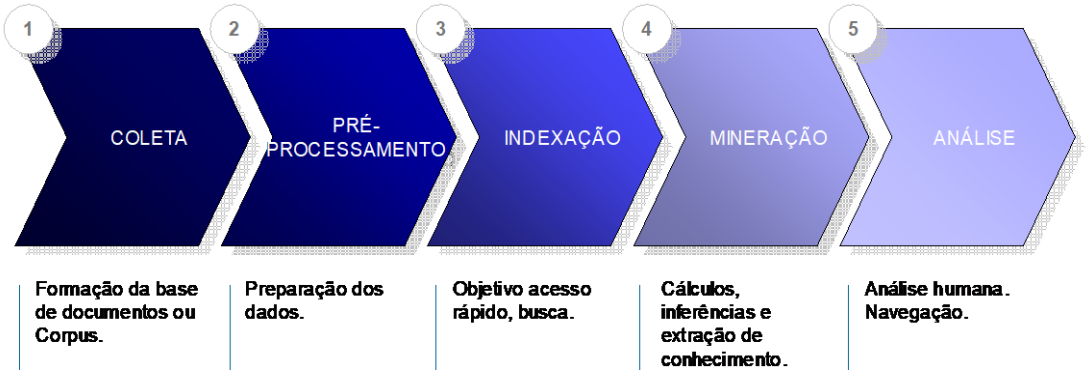

Figura 3: Etapas da Mineração de Texto
Fonte: Aranha (2007)

Em linhas gerais, a descrição dessa metodologia remete basicamente a um *modelo de aprendizado* que obtém conhecimento a partir de um *corpus textual*. A seguir, vejamos as principais definições de cada etapa.

3.2.1 Coleta de dados

Esta etapa é responsável pela seleção dos textos que irão compor a coleção de documentos. É importante lembrar que apenas *documentos inerentes ao domínio* do problema em questão devem ser considerados para a extração do conhecimento, uma vez que documentos fora do domínio podem impactar negativamente no processo de Mineração, além de aumentar a dimensionalidade dos dados (SOARES, 2008).

As coleções de documentos podem advir das mais diversas fontes, como: documentos pré-armazenados em alguma base de dados estruturada; ou fontes distribuídas sem qualquer estrutura de organização (GOMES, 2009). Algumas das principais fontes são: diretórios de arquivos de HDs[5], Bancos de Dados, Data Warehouses, textos eletrônicos digitalizados, e-mails e a Web em geral. Diversas

[5] Discos rígidos de computadores.

alternativas podem ser utilizadas para obtenção automática de dados, como *Crawlers*[6] e Sistemas de *Recuperação de Informação* (SRIs).

Uma vez que os dados tenham sido obtidos, cria-se um *corpus textual* que servirá de base para a aplicação das técnicas de mineração (BRITO, 2016). Vale ressaltar que o processo de criação desse corpus é demorado, principalmente pelo fato de exigir atividades manuais na maioria das situações com base no *julgamento de um especialista* (*expert judgment*) que possui conhecimento especializado sobre o domínio de interesse. (INDURKHYA; DAMERAU, 2010).

3.2.2 Pré-processamento

Em Mineração de Textos, os dados quase sempre estão num formato inadequado aos processos de descoberta de conhecimento. Dessa forma, é necessário que se apliquem algumas técnicas para seleção, transformação, limpeza e redução do volume destes dados, antes de serem submetidos à etapa de mineração (MORAIS; AMBRÓSIO, 2007).

De acordo com Gomes (2009), os sistemas de MT apenas submetem coleções de textos aos seus algoritmos se houver uma *preparação* prévia dos dados, denominada *Pré-processamento*. O objetivo dessa fase é permitir que os dados sejam transformados de maneira apropriada, retirando grande parte das anomalias, redundâncias e informações desnecessárias que podem acarretar resultados distorcidos (GONÇALVES, 2006). É de extrema importância que a estrutura resultante seja fidedigna à representação original dos dados. Tendo essa estrutura em mãos, é possível passar para a fase de Mineração, precedida ou não da fase de Indexação.

Carrilho (2007) também afirma que o pré-processamento de textos pode ser uma tarefa trabalhosa, principalmente pelo fato de não existir uma técnica ou um conjunto de técnicas gerais que garantam representações ideais de diversos domínios. Para esses casos, técnicas empíricas podem ser uma alternativa para uma melhor representação dos dados (ex.: pré-processamento manual). A seguir, são apresentadas algumas das principais tarefas utilizadas nessa fase.

[6] Robôs responsáveis por navegar sítios da Web e obter dela dados automaticamente, repassando-os para um componente de indexação das páginas obtidas (Gomes, 2009).

3.2.2.1 Tokenização

Segundo Soares (2008):

> O primeiro passo de uma operação de Pré-processamento é a tokenização ou atomização e sua execução tem como finalidade seccionar um documento textual em unidades mínimas, mas, que exprimem a mesma semântica original do texto (SOARES, 2008).

De acordo com a definição acima, a *tokenização*[7] visa desmembrar um texto em *tokens*, onde um *token* é a menor unidade de um texto que corresponde a uma palavra, na maioria das situações. Vale ressaltar que a separação das unidades de texto pode ocorrer em diferentes níveis, como capítulos, seções, parágrafos, frases, palavras e até sílabas (FELDMAN; SANGER, 2007). A criação de *tokens* por meio de delimitadores textuais é uma estratégia básica que apresenta bons resultados.

Exemplo:

Texto: *'This is a tokenization test.'*

Texto tokenizado: [‘This’, ‘is’, ‘a’, ‘tokenization’, ‘test’, ‘.’]

Em linhas gerais, o objetivo da *tokenização* consiste em obter um conjunto de dados estruturados a partir da transformação de um texto em dimensões passíveis de avaliação e análise (JACKSON; MOULINIER, 2007).

3.2.2.2 Case folding

O *case folding*, segundo Madeira (2015), é o procedimento que visa converter todos os caracteres de um documento para um único tamanho, seja maiúsculo ou minúsculo. Esse procedimento reduz consideravelmente a quantidade de atributos, porém é necessário ter o cuidado de não eliminar informações importantes para o problema em questão. Por exemplo, mensagens de *spam* geralmente vêm com letras em caixa alta. Assim, utilizar o *case folding* na filtragem de *spam* pode fazer com que informações importantes para caracterizar um *spam* sejam perdidas (SEGARAN, 2007 apud MADEIRA, 2015).

---

[7] Termo advindo do inglês, *tokenization*.

3.2.2.3 Remoção de *stopwords*

A *remoção de stopwords* consiste em eliminar palavras com pouco ou sem qualquer significado para um determinado contexto de aplicação de análise textual (FERREIRA, 2016). Alguns exemplos dessas palavras são: pronomes, preposições, artigos e advérbios (PASSINI, 2012). Abaixo, na Figura 4, é apresentado um exemplo o processo de *remoção de stopwords* a partir de uma *stoplist* de palavras.

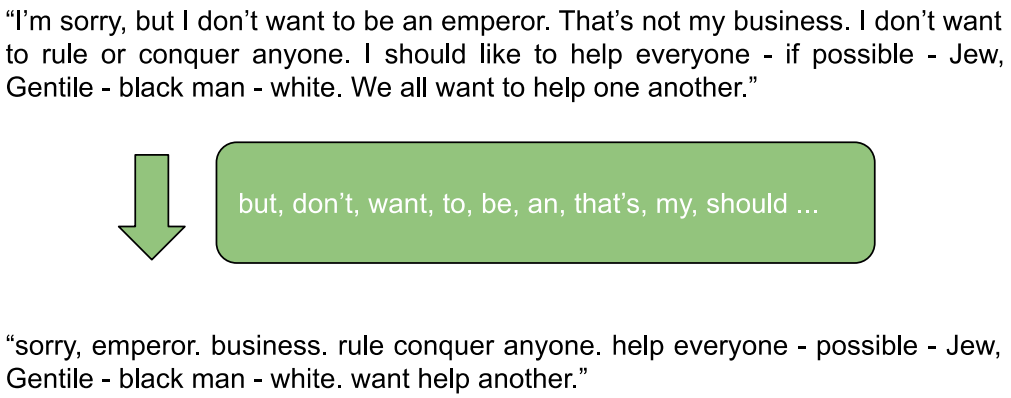

Figura 4: Processo de remoção de stopwords
Fonte: Autor (2018)

Geralmente, as *stopwords* são adicionadas manualmente a uma *stoplist*. No momento em que as tarefas de mineração são executadas, as palavras presentes nessa lista são desconsideradas. Vale ressaltar que essa etapa é opcional, uma vez que não é aplicável em todos os casos. Ao realizar uma consulta num *engenho de busca*, por exemplo, se a query for “*that* is” ou “that *are*” e o motor estiver configurado com uma *stoplist*, não haverá retorno, visto que todas as palavras da query estariam nessa *stoplist* (GOMES, 2009).

3.2.2.4 Radicalização da palavra

A *radicalização da palavra* (do inglês, *stemming*) tem como objetivo sintetizar uma palavra e identificar seu *lema* ou *radical*, eliminando os prefixos, sufixos, desinências, vogais temáticas (ANDRADE, 2016) e características de gênero, número e grau (MORAIS; AMBRÓSIO, 2007). Assim, palavras como *computer* e *computing* seriam ambas transformadas em *comput*, de modo que a representação final do texto terá um menor número de termos (PASSINI, 2012).

> [...] lema é a forma normalizada de uma palavra e, assim sendo, a lematização poderia ser explicada como a normalização de termos para um formato padrão da dimensão a qual os termos originais se aplicam (GOMES, 2009).

Existem, porém, alguns problemas na *stemmização* (Baeza-Yates e Ribeiro-Neto, 1999), como *Overstemming* (além do sufixo, parte do radical também é removida), *Understemming* (o sufixo não é removido ou é removido parcialmente) (PASSINI, 2012) e *palavras diferentes reduzidas à mesma palavra base (mesmo stem)* (GOMES, 2009). Diante disso, é necessário conhecer bem o domínio da aplicação e estar ciente dos objetivos que se deseja alcançar com o *stemming*.

Um dos algoritmos mais utilizados para *stemming* na língua inglesa para é o *Porter*. Ele é conceitualmente simples, dispondo de aproximadamente 60 sufixos, 2 regras de recodificação e um único tipo de regra sensível ao contexto para determinar se um sufixo deve ou não ser removido (WILLET, 2006). O *Porter* usa um comprimento mínimo baseado no número de *strings consoante-vogal-consoante* restantes após a remoção de um sufixo (WILLET, 2006).

3.2.2.5 Representação vetorial

Consiste em definir o conjunto de atributos/características que descrevem os documentos. Em alguns casos, cada atributo pode assumir apenas um valor *booleano*, indicando se um termo *x* do vocabulário está presente ou não no documento. Em outros casos, cada atributo corresponde a um peso numérico associado a um dado termo, o qual indica a importância do termo para o documento em questão. Nesse último caso, é muito comum a utilização do esquema *TF-IDF* (*Term Frequency – Inverse Document Frequency*), que é uma medida estatística para detectar a relevância de palavras em um documento (FELDMAN; SANGER, 2007). Vejamos a Equação 1 a seguir:

$$TF\text{-}IDF = \frac{Freq_{\mathrm{wd}}}{DocFreq_{\mathrm{w}}}$$ (Eq. 1), onde:

- $Freq_w$ é número de vezes que o termo *w* aparece no documento *d*;
- $DocFreq_w$ é número de documentos no qual o termo *w* aparece;

Em linhas gerais, o TF-IDF cria uma representação vetorial dos textos, que consiste num vocabulário contendo todos os tokens (i.e., palavras) do corpus. Um

documento, então, é expresso como sendo um vetor de números em que cada palavra do vocabulário presente no documento recebe um peso equivalente à sua importância nesse documento. Quanto mais rara a palavra for, maior será seu peso, e quanto menos rara, menor será o peso.

3.2.2.6 Seleção de atributos

A Seleção de atributos um procedimento que utiliza métricas para determinar o subconjunto mais representativo de um conjunto inicial de características, de modo a reduzir o espaço inicial. (CARRILHO, 2007). Este procedimento é muito útil, uma vez que dados textuais possuem altas dimensões (muitos *tokens*), contribuindo para a redução dos dados em dimensões menores (Redução da Dimensionalidade).

Essa redução também pode aumentar drasticamente a performance das tarefas de treinamento e classificação em modelos de *Aprendizagem de Máquina*. Várias métricas podem ser utilizadas para esse fim, como por exemplo *Informação Mútua* e X^2 (*qui-quadrático*), que são algumas das métricas estatísticas utilizadas no método de *Seleção univariada*. No contexto de *classificação textual*, tais métricas têm como objetivo avaliar a relação entre palavra e classe, ou seja, verificar o quão determinante ou não é uma palavra para uma dada classe. Vale lembrar que esse procedimento deve ser usado com cautela, para não haver grandes perdas de informação.

3.2.3 Indexação

A *indexação* consiste no processo através do qual as palavras de um texto são armazenadas em estruturas de índices, de modo a possibilitar a busca por documentos a partir das palavras neles contidas (SALTON; MCGILL, 1983 apud BARION; LAGO, 2008). As estruturas de índices são chamadas de *listas invertidas* ou *índices invertidos*. As *listas invertidas* têm o papel de indexar documentos através das palavras desses documentos, referenciando-os. Isso quer dizer que cada palavra é vista como um índice para todos os documentos que a contêm. Utilizam o mapeamento <*chave-valor*>, que nesse contexto as *chaves* são as palavras do vocabulário da base de documentos e os *valores* os conteúdos deles (BRITO, 2016).

O objetivo das listas invertidas é aprimorar o desempenho e a funcionalidade das buscas (ROCHA, 2002 apud BARION; LAGO, 2008), tal como o índice de um

livro, garantindo rapidez e agilidade (CASSENOTE, 2016) sem a necessidade de verificar sequencialmente a ocorrência de cada palavra em toda a base de documentos (MANNING; RAGHAVAN; SCHÜTZE, 2007 apud SOARES, 2013).

Também é importante destacar que a fase de indexação é diretamente influenciada pela fase de Pré-processamento, visto que todo o conteúdo que será indexado foi determinado por essa fase (SOARES, 2013).

3.2.4 Mineração

Essa é a mais importante etapa de todo o processo de Mineração de Textos, denominada também de *Extração de Padrões*. O objetivo dela consiste na busca efetiva por conhecimentos úteis a partir dos dados textuais. Esses conhecimentos são obtidos por meio da aplicação de algoritmos que produzem modelos de conhecimento capazes de realizar inferências sobre novos dados (PASSINI, 2012) (WITTEN; FRANK, 2005).

Também é nessa etapa que é selecionada a tarefa/técnica de mineração que melhor se aplica ao problema em questão. Por exemplo, caso o objetivo da mineração seja a avaliação do grau de similaridade existente entre um grupo de documentos textuais, então estamos lidando com uma tarefa de *clusterização*. Em outro exemplo, caso queiramos classificar um conjunto de documentos quanto aos assuntos relacionados a eles, estamos nos referindo à tarefa de *classificação*.

Vale ressaltar também que, dependendo da natureza do problema, diversas tarefas/técnicas podem ser empregadas em conjunto para melhor atingir os objetivos propostos (BRITO, 2016). Para este trabalho em particular, foram consideradas as técnicas de *classificação*.

3.2.5 Análise

Segundo Soares (2013), esta etapa diz respeito aos procedimentos posteriores que deverão ser realizados sobre o conhecimento extraído na etapa de *Mineração*, consistindo em *análise*, *visualização* e/ou *interpretação* desse conhecimento. Esses procedimentos visam facilitar e viabilizar a compreensão e a interpretação do conhecimento descoberto, bem como avaliar sua utilidade (PASSINI, 2012). Brito (2016) afirma que essa etapa também é responsável pela validação da eficiência do

processo como um todo, ou seja, avaliar se o objetivo de descobrir um novo conhecimento foi de fato alcançado.

Carvalho (2017) afirma que o conhecimento obtido deve ser avaliado e submetido a métricas de qualidade e desempenho, a fim de selecionar o que é importante, relevante ou interessante ao usuário. Caso se perceba que o conhecimento obtido é insatisfatório, todo o processo terá que ser repetido, aplicando-se outras técnicas de modo a alcançar os objetivos inicialmente propostos (CARVALHO, 2017). Madeira (2015) afirma ainda que há duas formas de se avaliar os resultados obtidos: *forma subjetiva*, baseada no conhecimento de um especialista de domínio; ou *forma objetiva*, através de métricas que calculam a qualidade dos resultados.

Os resultados dos processos de Mineração de Dados normalmente são analisados através de métricas de desempenho (*forma objetiva*) (Soares, 2013). As principais métricas de avaliação utilizadas em MT são advindas da área de RI, seguindo o princípio de *relevância*. Um texto é *relevante* caso tenha importância para um dado tópico. A *Precisão*, a *Cobertura* e a *F1-score* são as métricas de desempenho mais utilizadas em MT (ver detalhes na seção 3.4). Porém, de acordo com o objetivo de cada processo, métricas de avaliação de desempenho diferentes das citadas acima podem ser utilizadas.

3.3 APRENDIZAGEM DE MÁQUINA

Inicialmente, vale destacar duas abordagens em Aprendizagem de Máquina (AM): *aprendizado supervisionado*, que dispõe de um conjunto de dados etiquetados; e *aprendizado não supervisionado*, onde não há etiquetagem do conjunto de dados. As tarefas mais comuns do aprendizado supervisionado são a *classificação* e a *regressão*, enquanto a tarefa mais comum do aprendizado não supervisionado é o *agrupamento* (*clustering*) (MCCUE, 2007 apud CAMILO; SILVA, 2009). Dentro do escopo desta dissertação, utilizamos apenas o *aprendizado supervisionado*, pois o contexto do problema (capítulo 4) remete a um problema claro de *classificação*.

Na *aprendizagem supervisionada*, existem diversos conjuntos de dados, onde cada instância de dado possui uma variável-alvo pré-definida (target). Para atribuir a variável-alvo correspondente para uma nova instância, um algoritmo de AM realiza uma série de procedimentos, como indução de regras e cálculos probabilísticos, a

partir das características observadas nos conjuntos de dados etiquetados inicialmente fornecidos. Assim, os algoritmos conseguem “aprender” as características gerais desses dados e realizar poderosas inferências sobre novos dados (ex.: atribuição da target correta para uma nova instância, em um problema de *classificação*). O algoritmo treinado é então “persistido” em um *modelo,* que poderá ser utilizado futuramente em novas amostras de dados, isto é, que não foram previamente etiquetadas.

Essa abordagem tem dois ciclos distintos: *criação/indução* e *uso* do modelo. O ciclo de criação do modelo é subdividido em duas fases: treinamento e testes. Tão logo o processo de aprendizado (indução) estiver concluído, o modelo deve ser avaliado, de modo a verificar estatisticamente a qualidade dos resultados alcançados. Essa avaliação deve utilizar dados novos, isto é, dados ainda não conhecidos pelo modelo. O uso de dados novos nessa etapa contribui para uma avaliação honesta e realista do desempenho do modelo criado, uma vez que esses dados não estavam presentes no processo de treinamento. Por isso é necessário que haja uma divisão dos dados em *treinamento* e *teste* (DAMASCENO, 2010).

Existem casos em que dados são divididos em 3 categorias: *treinamento*, *teste* e *validação*. Os dados de *validação* servem para ajustar os parâmetros do modelo e para aumentar sua capacidade de inferência para dados desconhecidos. Em termos percentuais, a divisão de dados em *treinamento* e *teste*, resulta em 70% e 30% respectivamente; já a divisão em *treinamento*, *teste* e *validação*, resulta em 70%, 20% e 10% respectivamente (DAMASCENO, 2010).

No contexto da *Mineração de dados* convencional, Damasceno (2010) afirma que:

> Toda técnica de mineração passa por um processo chamado de treinamento. A fase de treinamento tem este nome por ser um processo de apresentação dos dados processados para o algoritmo de mineração, cujo objetivo é identificar, ou seja, “aprender” as características ou padrões úteis ao objetivo do processo de descoberta de conhecimento (DAMASCENO, 2010).

Outro aspecto que precisa ser ressaltado é a possibilidade de criar algoritmos inteligentes utilizando apenas *abordagens baseadas em regras* (do inglês, rules-based approaches). Tais algoritmos compreendem um conjunto de regras de formato IF-THEN, em que uma dada regra *k* “cobre” uma instância *x* se os atributos a_i de *x* satisfazem *k* (CAMILO; SILVA, 2009; SHOARAN, 2015).

3.4 CLASSIFICAÇÃO DE TEXTOS

O crescimento e a disponibilidade de documentos na internet têm ampliado fortemente o interesse em estudos na área de MT, sobretudo nas técnicas de *classificação* de textos (Brito, 2016). Como já mencionado na seção anterior, classificadores podem ser criados usando-se técnicas baseadas em Regras e Aprendizagem de Máquina.

Segundo Rodrigues (2009), a *classificação de texto* consiste em

> [...] associar documentos de texto a classes temáticas pré-definidas. Por exemplo, notícias jornalísticas podem ser associadas a categorias de política, esportes, entretenimento, dentre outras. Ou ainda, mensagens de e-mail podem ser classificadas como spam ou não-spam, ou um paciente como doente ou sadio, e assim por diante (RODRIGUES, 2009).

A *classificação de texto*, portanto, consiste em atribuir classes (ou rótulos) a documentos textuais (BRITO, 2016).

Sebastiani (2002) afirma que:

> [...] a *classificação de textos* consiste em determinar se um documento d_i, (de um conjunto de documentos D) é pertencente ou não a uma categoria c_j (de um conjunto de categorias C), consistentemente com o conhecimento das categorias corretas para um conjunto de documentos de treinamento (SEBASTIANI, 2002).

3.4.1 Definição do problema de classificação textual

O problema de *classificação* pode ser compreendido como sendo “a atribuição de classes a documentos de texto $V = (V_1, V_2, ..., V_l)$.”. Nos casos em que um documento é associado apenas a uma categoria, estamos lidando com um problema de classificação *single-label*, tal como uma mensagem de e-mail classificada como *spam* ou *não-spam*. Também existem os casos que um documento pode ser classificado em mais de uma categoria, sendo então um problema *multi-label*, como por exemplo o diagnóstico de um paciente ser classificado em mais de um tipo de doença (RODRIGUES, 2009) (KORDE; MAHENDER, 2012).

Dentro da abordagem de AM, Sebastiani (2002) afirma que os classificadores são criados a partir de um processo *indutivo* de aprendizado e de relação entre as características e as *etiquetas* (classes) dos documentos utilizados. A tarefa começa

com um conjunto de documentos de treinamento $D = (d_1, d_2, ..., d_n)$ etiquetados (com classes A, B ou C, por exemplo). O objetivo é então determinar um modelo de classificação capaz de atribuir a classe correta a um novo documento *d*. (KORDE; MAHENDER, 2012).

As fases a seguir definem um modelo de classificação textual baseado em algoritmos de AM, bem como o diagrama das fases na Figura 5, de acordo com Rodrigues (2009):

(1) Obtenção dos documentos de um determinado domínio (documentos do conjunto *D*);

(2) Criação do vocabulário $(w_1, ..., w_i)$ de *i* termos, a partir de corpus de documentos. Envolve *pré-processamento textual* e *transformação dos dados* (representação vetorial dos textos, TF-IDF);

(3) Redução da dimensionalidade, para seleção dos atributos/características mais relevantes da representação inicial ($M < T$);

(4) Indução do classificador a partir do conjunto de treinamento (i.e., dados transformados);

(5) Avaliação do desempenho do classificador, dado um conjunto de testes. Diversas métricas e técnicas podem ser utilizadas para avaliação, sendo as mais comuns:

(a) *Precisão*: número de documentos associados corretamente pelo classificador a uma dada classe dividido pelo número de documentos associados pelo classificador à classe;

(b) *Cobertura*: número de documentos associados corretamente pelo classificador a uma dada classe dividido pelo número de documentos existentes da classe;

(c) *Validação cruzada* (*Cross-validation*): técnica para avaliar se um modelo possui um poder de generalização aceitável, ou seja, se ele está apto a classificar corretamente novas instâncias. Consiste em desmembrar todo o conjunto de treinamento em *k* partes. Uma das partes é reservada para *teste* e as demais para *treinamento*, e isso é feito iterativamente para que o modelo seja treinado/testado com todos as *k* partes (SANTANA, 2018).

(d) *F1-score*: Média harmônica entre *precisão* e *cobertura* para medir a performance do classificador, dando uma visão mais exata da eficiência

de um modelo que o uso das duas métricas em separado.

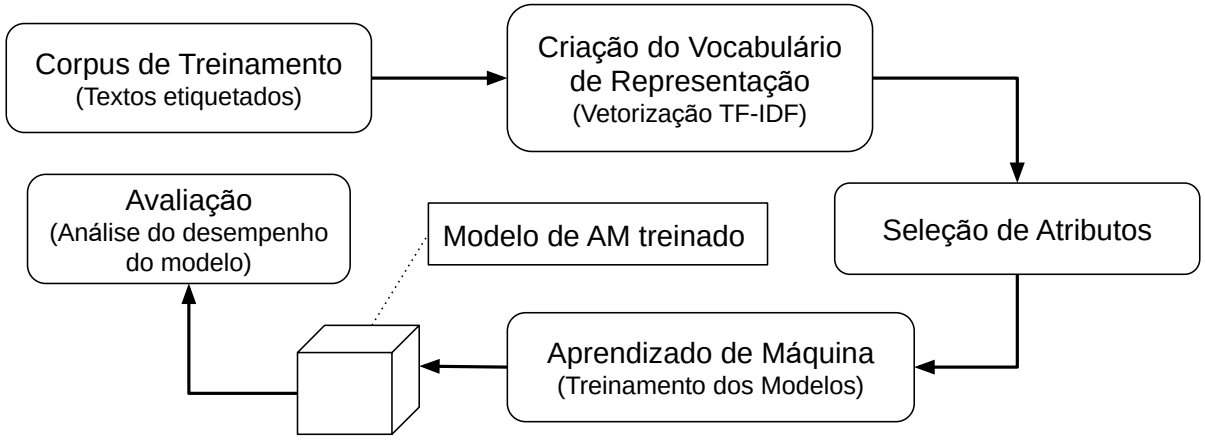

Figura 5: Fases da classificação de textos
Fonte: Adaptado de Rodrigues (2009)

3.4.2 Algoritmos de AM para Classificação de Texto

Para este trabalho, foram contemplados 3 algoritmos comumente utilizados em classificação de texto. São eles: *Naïve Bayes*, *Support Vector Machines* e *K-Nearest Neighbors*. A seguir, temos uma breve descrição de cada um deles, suas principais características, seus potenciais e suas deficiências.

3.4.2.1 Naïve Bayes (NB)

Algoritmo probabilístico baseado no teorema de *Bayes*. Ele calcula a probabilidade $P(c \mid d)$ de um documento pertencer a uma determinada classe, dada a probabilidade inicial $P(c)$ de um documento pertencer a essa classe e das probabilidades condicionais $P(t_k \mid c)$ de cada termo t_k ocorrer em um documento da mesma classe (LUCCA et al., 2013). O objetivo do NB é determinar a classe C_{map} mais provável de uma nova instância a partir dos atributos que descrevem essa instância (MITCHELL, 1997 apud RODRIGUES, 2009). Na Equação 2 temos o cálculo dessa classe mais provável.

$$C_{\text{map}} = \arg max_{c \in C} P(c|d) = \arg max_{c \in C} P(c) \prod_{1 \le k \le nd} P(tk|c)$$

(Eq. 2)

A vantagem do Naïve Bayes é a simplicidade do seu cálculo, pois lida "ingenuamente" com os atributos das instâncias como sendo independentes, de modo a obter a classe que maximize o resultado de sua fórmula (MITCHELL, 1997 apud BORGES, 2006). O NB também tem apresentado bons resultados em vários problemas de alta complexidade, principalmente os de *classificação de textos* (FRANK; BOUCKAERT, 2006 apud RODRIGUES, 2009).

Algumas das principais características do NB segundo Rodrigues (2009) são: utiliza baixo custo computacional; não gera regras explícitas; é instável em relação às instâncias de treinamento; é sensível à redundância nos atributos e também a classes balanceadas; não é sensível a atributos irrelevantes.

Borges (2006) afirma que estudos comparativos têm demonstrado que os algoritmos bayesianos, em alguns casos, podem apresentar bons resultados de desempenho, principalmente se comparados a técnicas como *árvore de decisão* e *redes neurais* e também utilizando métodos para seleção de atributos e remoção de redundância nos dados.

3.4.2.2 Support Vector Machines (SVM)

Algoritmo discriminativo formalmente definido por um *hiperplano de separação*, ou seja, dadas as instâncias etiquetadas de treinamento, é gerado um *hiperplano ótimo* capaz de classificar novas instâncias. Se considerarmos um espaço de duas dimensões, esse hiperplano é representado por uma linha que separa um plano em duas partes, onde cada classe fica com uma dessas partes (PATEL, 2017).

Para Rodrigues (2009), o SVM baseia-se na ideia de encontrar um *hiperplano ideal* que separe dois conjuntos de pontos linearmente separáveis no espaço. Tão logo o hiperplano é encontrado, basta, para realizar a classificação de uma nova instância (ponto), verificando em qual área (lado esquerdo ou direito) deste hiperplano ela se encontra. Assim, conclui-se que o SVM consiste numa *abordagem geométrica* para o problema de classificação, onde cada uma das instâncias de treinamento é vista como um ponto x_i num espaço R^M, e o aprendizado dá-se pela divisão de elementos *positivos* e *negativos* neste espaço (OGURI, 2006).

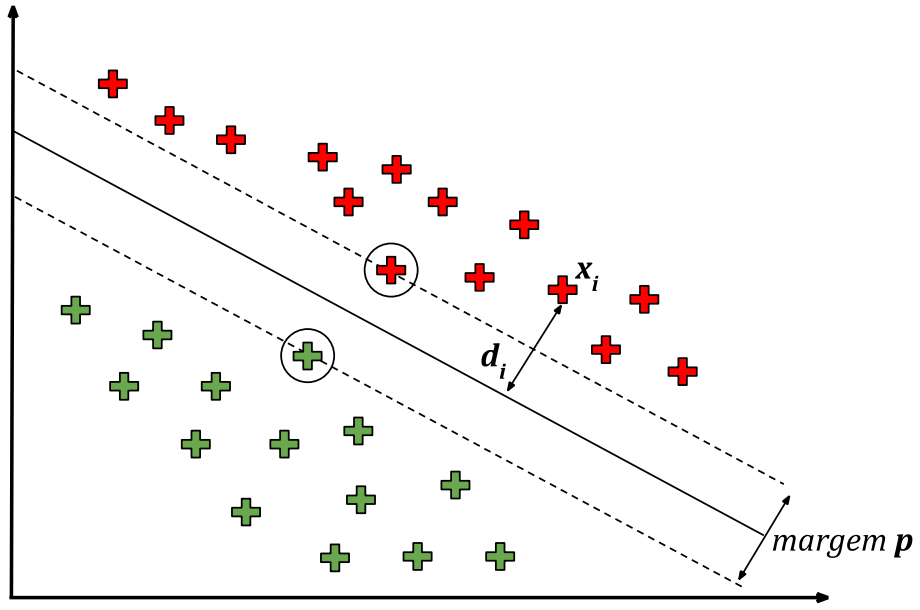

Figura 6: Margem geométrica de um ponto x_i e a margem p do hiperplano ótimo
Fonte: Adaptado de Borges (2006)

Vale ressaltar que podem existir infinitos hiperplanos que separam dois conjuntos de pontos linearmente separáveis. O desafio do SVM é encontrar o hiperplano ótimo que maximize a *margem p* (distância entre dois elementos de classes diferentes) para os pontos mais próximos (*vetores de suporte*[8]) a ele (Figura 6), diminuindo assim as chances de classificar novos pontos de forma equivocada (RODRIGUES, 2009). Na ilustração da Figura 5, as linhas tracejadas são os vetores de suporte para essas instâncias de treinamento, e cada instância que representa um vetor de suporte está circundada por um círculo mais externo (LIMA, 2002 apud BORGES, 2006).

Apesar da ideia do SVM ser melhor compreendida nas situações em que os pontos positivos e negativos são linearmente separáveis, ele também aplicável em casos opostos (não linearmente separáveis) (SEBASTIANI, 2002). Para isso, o SVM utiliza uma função de *kernel*. Segundo RODRIGUES (2009), essa função tenta mapear os pontos originais para um novo espaço onde eles sejam linearmente separáveis, e encontrar o hiperplano ótimo em seguida. Várias funções de *kernel* têm sido usadas em SVMs, como funções polinomiais (lineares e quadráticas), e funções gaussianas (RODRIGUES, 20019) (BORGES, 2006).

[8] Vetores de Suporte (do inglês, Support Vectors) são padrões que representam as instâncias de treinamento mais próximos ao hiperplano ótimo e caracterizam-se por serem um pequeno conjunto dos dados mais informativos dessas instâncias (SOUTO et al., 2003 apud BORGES, 2006). Assim, o classificador SVM realiza as inferências por meio deles (SEBASTIANI, 2002).

Quanto às principais características do SVM, podemos pontuar, segundo Rodrigues (2009): alto custo computacional; não gera regras explícitas; exige definição de parâmetros; estável em relação às instâncias de treinamento; pouca sensibilidade à atributos redundantes; sensível a classes desbalanceadas.

3.4.2.3 K-Nearest Neighbors (KNN)

De acordo com Borges (2006), o KNN é um algoritmo baseado em instâncias, que são agrupadas de acordo com a maior proximidade que elas têm entre si. O aprendizado deste algoritmo é considerado *tardio*, pois a classificação de novas instâncias acontece sob demanda, necessitando que os dados de treinamento sejam armazenados e recuperados à medida que for solicitada uma classificação para uma nova instância, porém permite um aprendizado de forma incremental (CAMILO; SILVA, 2009).

O KNN considera que as instâncias são representadas por pontos num espaço de dimensão *n*. Para classificar uma nova instância *x*, o KNN imputa a *x* a classe mais frequente dentre as *k* instâncias de treinamento mais próximas, ou seja, que possuam a menor distância de *x* (RODRIGUES, 2009). Os dois principais pontos-chave do KNN são: a métrica de distância e o valor de *k*. A variável *k* determina a quantidade de instâncias mais próximas que serão usadas para verificar à qual classe a nova instância pertence. (PACHECO, 2017). A distância é calculada através de funções de similaridade, como *distância euclidiana* e *cosseno* (KORDE; MAHENDER, 2012).

Para exemplificar, consideremos a *distância euclidiana* para demonstração de como a distância entre uma instância *x* e as instâncias de treinamento é calculada (BORGES, 2006): uma instância *x* é representada pelo vetor de atributos: $a_1(x)$, $a_2(x)$, …, $a_r(x)$, onde $a_r(x)$ representa o valor do *r*-ésimo atributo da instância *x*. Logo, a distância entre duas instâncias x_i e x_j é definida como $d(x_i, x_j)$, em que (Equação 3):

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} [a_r(x_i) - a_r(x_j)]^2} \quad \text{(Eq. 3)}$$

Os problemas mais comuns do KNN estão relacionados com: a definição de um *k* ótimo e o uso de todos os atributos das instâncias de treinamento para a computação das distâncias (PACHECO, 2017) (KORDE; MAHENDER, 2012). Em

relação à constante *k*, não há um valor padrão a ser sempre atribuído, pois esse valor pode variar conforme a base de dados. Dependendo do problema que se queira resolver, pode ser usado um *algoritmo de otimização*[9] para encontrar o melhor valor de *k*, embora isso venha a impactar a performance do algoritmo no momento de determinar *k*. Outra alternativa é utilizar o método de *tentativa-erro* de modo a encontrar *k* empiricamente. Para esse caso, é aconselhável utilizar valores ímpares/primos (PACHECO, 2017). Em relação à distância, computá-la é tarefa custosa, principalmente se o problema em questão possuir um número grande de instâncias, levando o algoritmo a consumir um tempo de cálculo muito longo (PACHECO, 2017).

No campo da classificação textual, na grande maioria dos conjuntos de dados textuais, apenas uma pequena parcela do vocabulário total é útil para determinar a classe de novas instâncias. Uma abordagem que pode resolver tal problema é aprender pesos para diferentes atributos (i.e., termo/palavras dos documentos) (KORDE; MAHENDER, 2012).

Finalmente, as principais características do KNN são de acordo com Rodrigues (2009): alto custo computacional; não gera regras explícitas; requer definição de parâmetros (ex.: valor de *k*); estável em relação aos dados de treinamento; sensível a atributos redundantes e irrelevantes; sensível a classes desbalanceadas.

[9] Em linhas gerais, *otimização* objetiva determinar pontos extremos de uma função matemática (*aptidão ou fitness*), seja ele mínimo ou máximo, para problemas de minimização e maximização respectivamente (PACHECO, 2017).

3.5 CONSIDERAÇÕES FINAIS

A MT oferece uma abordagem poderosa e eficiente para a obtenção de conhecimento em dados textuais. Diferentemente da MD convencional, seu objetivo consiste em lidar com dados não estruturados. Esses dados, no entanto, precisam ser pré-processados, para estarem aptos a serem submetidos aos algoritmos de mineração. Tais algoritmos utilizam técnicas já consagradas da Aprendizagem de Máquina, que provê meios para lidar com os dados e suas complexidades e garante uma boa acurácia dos resultados.

Das técnicas utilizadas para lidar com dados textuais, a *classificação de texto* baseada em AM configura uma das principais, utilizada sobretudo para identificar a classe adequada para um documento textual a partir de outros dados textuais de treinamento. O modelo induzido identifica as características mais representativas do conjunto de treinamento e as utiliza para classificar as novas instâncias. Existem diversos algoritmos de classificação, sendo destacados neste trabalho, especificamente, o Naïve Bayes, o K-Nearest Neighbors e o Support Vector Machines. Na literatura, esses algoritmos são alguns dos mais utilizados para classificação textual. Cada um deles, como citado anteriormente, possui vantagens e desvantagens, podendo apresentar melhores resultados para um dado problema, e resultados insatisfatórios em outros casos.

É importante considerar não apenas os algoritmos em si, mas outros fatores, como o conjunto de dados, as tarefas de pré-processamento, a seleção de atributos e outros aspectos que também são responsáveis pela qualidade dos resultados da classificação, sendo necessária a realização de eventuais ajustes, de modo a prover uma melhor configuração do modelo de classificação e ampliar sua acurácia.

4 DOGFOODING ANALYSIS SYSTEM

Este capítulo tem por objetivo apresentar a descrição do trabalho realizado, contando com três seções: (4.1) Problema identificado, (4.2) Solução proposta e (4.3) Considerações finais. A seção 4.2 apresenta o detalhamento dos módulos, que resumem as principais funcionalidades do protótipo.

4.1 PROBLEMA IDENTIFICADO

Como já foi mencionado no capítulo 2 deste trabalho, *dogfooding* é uma abordagem característica de empresas que usam os próprios produtos, com objetivo de atestar o seu grau de confiança em relação ao que produzem. Este trabalho foi desenvolvido no contexto de uma colaboração entre o CIn-UFPE e a Motorola Mobility. A Motorola possui uma política interna de dogfooding. Essa política consiste em disponibilizar protótipos de aparelhos para vários colaboradores, que fornecem feedbacks de elogios, sugestões, críticas e de relatos sobre defeitos de software pertinentes aos aparelhos em comunidades privadas de dogfooding da própria empresa. A equipe de TEs realiza análises frequentes sobre esses feedbacks, com o objetivo de obter informações úteis que podem ser utilizadas como diretrizes extras de teste, para detecção de bugs não previstos.

No entanto, as comunidades internas não possuem mecanismos para identificar os feedbacks uteis (i.e., relevantes). A ausência dessa identificação faz com que os testadores gastem um certo tempo para analisar cada feedback a fim capturar os que são úteis ao planejamento. Nesse processo, alguns dos feedbacks úteis não são contemplados na análise, devido à dificuldade e o tempo gasto para identificar todos eles. Além disso, os testadores também precisam, manualmente, capturar os links dos feedbacks relevantes, mapear os charters relacionados a esses feedbacks e reportar a análise realizada no sistema interno de gerenciamento de tarefas (Atlassian Jira[10]), de acordo com a política adotada.

[10] © Atlassian Jira 2018

4.2 SOLUÇÃO PROPOSTA

Considerando os problemas mencionados na seção anterior, foi sugerida uma solução computacional capaz de automatizar boa parte dos procedimentos manuais pertinentes às atividades de análises de feedbacks de dogfooding. Esta seção apresenta a visão geral e o processo proposto da solução criada, descrevendo os módulos principais.

4.2.1 Visão Geral

Buscando automatizar boa parte dos procedimentos manuais durante as análises de DF, foi elaborado um processo cujo objetivo é realizar as seguintes tarefas: classificar automaticamente os feedbacks de *dogfooding;* recuperar aqueles que contêm cenários de defeitos, e que, portanto, são úteis ao planejamento de TEs (feedbacks relevantes); mapear charters relacionados às áreas mencionadas nesses cenários; e permitir submissão semiautomática da s análises.

Esse processo foi implementado em um protótipo de um sistema computacional, apelidado de *"DAs"* (*Dogfooding Analysis system*), que utiliza conceitos de *Aprendizagem de Máquina* aplicados a textos e *Recuperação de Informação*. Essa solução teve como foco proporcionar um ganho significativo de produtividade quanto à realização das análises de DF, isto é, ganho de tempo e identificação de feedbacks relevantes não mapeados manualmente.

A ideia da proposta surgiu com base nas atividades manuais realizadas pela equipe de testes exploratórios da organização, que consistem em: (1) consultar as comunidades de dogfooding de cada aparelho a testar; (2) buscar por feedbacks relevantes; (3) listar feedbacks relevantes; (4) associar charters aos feedbacks relevantes; (5) criar documento de análise; (6) submeter análise.

4.2.2 Processo Geral e Descrição dos Módulos

A Figura 7 apresenta o processo geral do protótipo do sistema considerando os módulos principais: (1) obtenção dos feedbacks a analisar; (2) classificação dos feedbacks; (3) busca e seleção dos charters relacionados; (4) criação e submissão da análise de dogfooding.

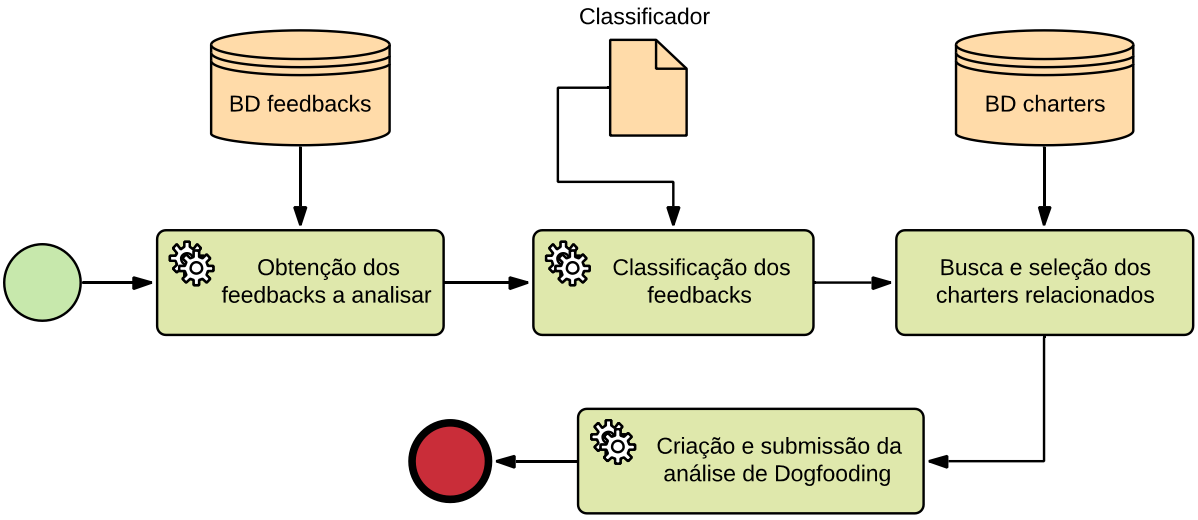

Figura 7: Diagrama BPMN do processo geral do protótipo
Fonte: Autor (2018)

4.2.2.1 Obtenção dos feedbacks a analisar

Este módulo consiste em prover um método para que os testadores possam obter os feedbacks dos aparelhos para realizar as análises. No uso das comunidades de dogfooding, os testadores precisam fazer solicitações e aguardarem aprovação dos moderadores para poder ter acesso aos feedbacks. Cada aparelho possui uma comunidade particular, então os testadores precisam solicitar participação em cada uma delas.

No uso do protótipo, a obtenção é feita por meio de consultas SQL. O protótipo dispõe de uma base de dados criada exclusivamente para manter os dados dos feedbacks de dogfooding de todos os aparelhos atuais em teste. Na interface do protótipo, basta que o usuário selecione o *aparelho*, *o intervalo de data* (i.e., para recuperar os feedbacks com data de postagem dentro desse intervalo), *a número de feedbacks* a recuperar e em seguida acione o botão de *"busca"* (lupa). Esses são os três parâmetros passados internamente através de consultas SQL na base de dados de feedbacks. Vale ressaltar que, caso o testador não informe o *número de feedbacks*, o protótipo irá recuperar 10 feedbacks por padrão. Esse número foi definido de acordo com os testadores. Vejamos então um exemplo de obtenção dos feedbacks nas Figuras 8 e 9, que mostram a GUI do protótipo.

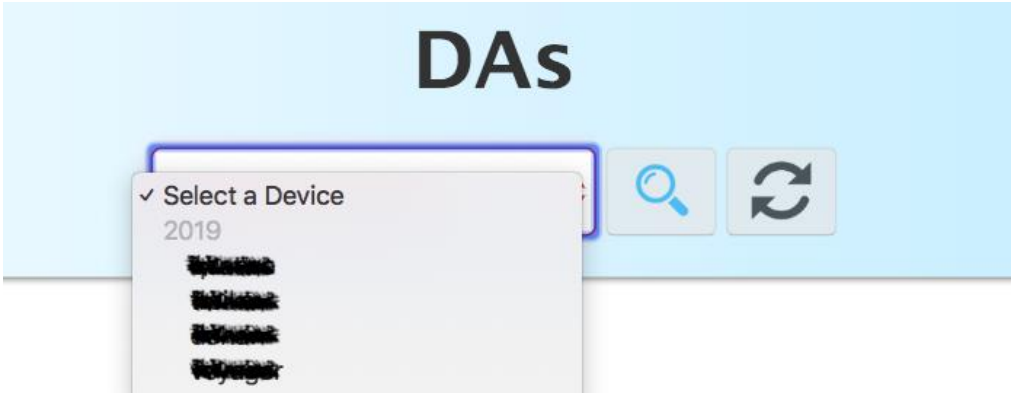

Figura 8: DAs – Seleção do aparelho para recuperar seus feedbacks
Fonte: Autor (2018)

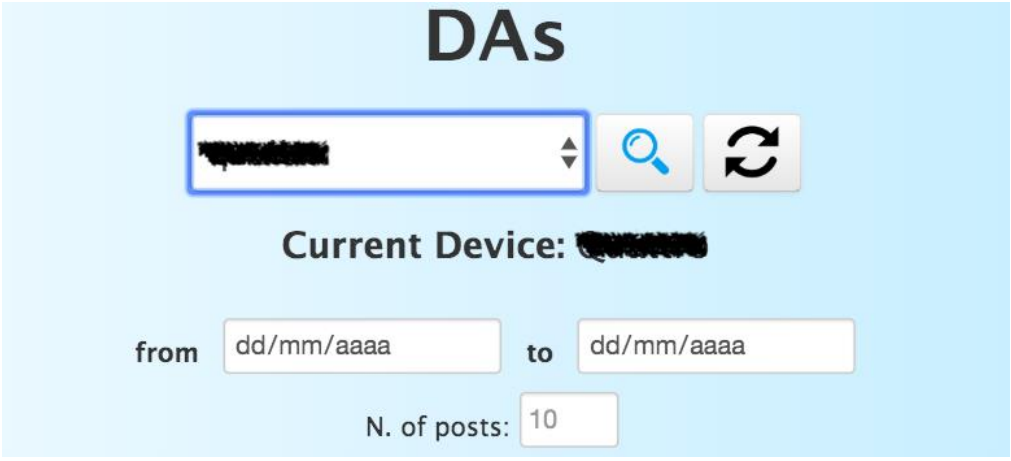

Figura 9: DAs – Definições do intervalo de data e número de feedbacks a recuperar
Fonte: Autor (2018)

Na Figura 8, vemos que os aparelhos estão listados pelo seu ano. Cada ano contém uma lista de aparelhos. O protótipo considera os aparelhos do ano corrente e de até 2 anos atrás. Esses aparelhos mais antigos estão inclusos devido às atualizações de software que podem ocorrer para alguns deles, de acordo com os testadores. Porém, para cada atualização, uma nova comunidade de dogfooding é criada. A inserção dos dados dos aparelhos e de seus feedbacks é realizada periodicamente por colaboradores de outros países, os quais extraem esses dados das comunidades e os armazena na base de dados de onde o protótipo fará o acesso.

O intervalo de data é definido por dois campos: *from*, que é a data passada e *to*, que é a data mais recente. Um exemplo de intervalo seria: *from 05/05/2018 to 09/05/2018*. Por *default*, caso o intervalo não seja informado, feedbacks de até 4 dias (contando com o dia corrente) são recuperados. Esse intervalo *default* normalmente é utilizado pelos testadores nas atividades de início de semana (i.e., na segunda-

feira), porque vários colaboradores postam feedbacks todos os dias, inclusive nos finais de semana. Dessa forma, os testadores precisam obter os feedbacks desde a sexta-feira da semana anterior até segunda-feira da semana corrente. A partir da terça-feira da semana corrente, a recuperação irá considerar apenas os feedbacks do dia anterior mais os feedbacks do dia corrente (i.e., os feedbacks da segunda-feira e da terça-feira, por exemplo). Também vale ressaltar que, dos *N* feedbacks a recuperar, são retornados *M* < *N* feedbacks, pois só os que foram classificados como relevantes são retornados ao testador.

4.2.2.2 Classificação dos feedbacks

Este módulo é responsável por realizar a classificação dos feedbacks para um dado aparelho. Quando o testador realiza o procedimento do módulo anterior e aciona o botão de “busca”, os feedbacks são recuperados, tratados, transformados para um formato estruturado e lançados como entrada para um *classificador* baseado em *Aprendizagem de Máquina*. O classificador, então, atribui *labels* (i.e., relevante ou irrelevante) correspondentes a esses feedbacks, de acordo com os dados que foram passados para treiná-lo. Ao final, o protótipo apresentará ao testador uma lista dos feedbacks classificados como *relevantes*, os quais possuem informações que podem ajudar a direcionar os testes exploratórios, isto é, ajudar a detectar novos defeitos relacionados aos defeitos mencionados nos feedbacks. Vejamos a Figura 10 a seguir, que mostra a tela do protótipo com um exemplo de um feedback classificado.

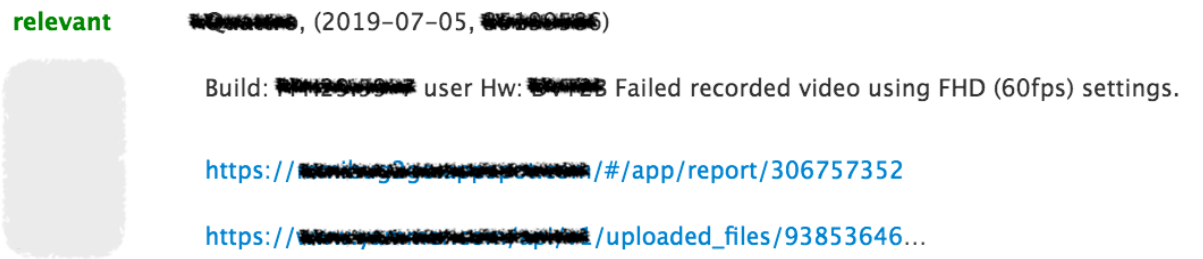

Figura 10: DAs – Feedback classificado
Fonte: Autor (2018)

Vemos que o texto do feedback da figura acima refere-se a um comportamento defeituoso. O protótipo retorna os feedbacks contendo: o texto, a *label* de classificação (label *relevant*) e alguns links com informações adicionais sobre o feedback (imagens, logs de defeito, etc.). Essas informações (com exceção das *labels* de classificação)

são fornecidas pelos *dogfooders* quando estes fazem as suas postagens dos feedbacks na comunidade de dogfooding correspondente ao aparelho.

Como foi visto na seção 3.3, um modelo de classificação é criado a partir de um conjunto de amostras de dados etiquetados, que nesse caso são os *feedbacks* de dogfooding etiquetados. Este problema de classificação, em particular, caracteriza-se como um problema *binário*, onde são consideradas duas classes para categorizar o conjunto de dados: classe *relevante* (quando um feedback faz menção a defeitos) e classe *não-relevante* (quando o feedback não se refere a defeitos), sendo que apenas uma delas é útil.

A etiquetagem dos dados foi realizada manualmente por um dos colaboradores do time de TEs da organização, através de análises dos textos dos feedbacks. Nesse processo, houve alguns fatores que ajudaram a qualificar mais precisamente um feedback. Por exemplo, sentenças como “Phone app crash”, “Photo quality is strange” e “I think I’ve a problem on my device speaker”, são algumas das características dos *feedbacks relevantes*, pois estão referindo-se a defeitos e situações possivelmente problemáticas. Já outras sentenças, como “Camera is cool”, “Good device design” e “Screen size is ideal”, são características de *feedbacks irrelevantes*, pois não agregam valor às atividades de testes de software. A etiquetagem manual considerou essas e outras características semelhantes nos feedbacks, para então categorizá-los corretamente.

Os feedbacks etiquetados, tal como os novos feedbacks que os testadores desejam classificar, também precisam ser tratados e transformados para um formato estruturado antes de serem lançados como entrada para o modelo de AM. Ao final, o modelo é treinado com esses feedbacks e é em seguida persistido, para ser utilizado na classificação dos novos feedbacks. Este módulo, portanto, compreende várias tarefas. No entanto, em termos de uso, o testador utiliza apenas classificação de feedbacks e a indicação dos feedbacks corretamente classificados. A ilustração da Figura 7 apresenta o fluxo com o modelo treinado e disponível para uso.

Vale ressaltar que a motivação para utilizar AM neste trabalho partiu de que nem sempre os feedbacks são óbvios em relação à sua classe. Por exemplo: “In my phone always appears update message, even though I have already updated”. Vemos nesse exemplo que, apesar do feedback aparentar uma situação de defeito, não há um termo necessariamente negativo para dizer que de fato trata-se de um defeito. No começo da pesquisa, foram criados alguns algoritmos baseados em regras com

palavras de *blacklist*, que remetem a defeitos mais comuns. No entanto, a quantidade de regras a serem criadas seria muito alta, em função de situações como essas. Dessa forma, tais regras poderiam acarretar num número de falsos positivos e negativos mais alto. Então, foi pensada a possibilidade de utilizar AM.

4.2.2.3 Módulo de Busca e Seleção dos Charters relacionados

Além de retornar os feedbacks relevantes para o testador, o protótipo também busca e disponibiliza uma lista dos possíveis *charters* relacionados a cada feedback, podendo ser selecionados um ou mais charters.

Como dito no capítulo 2, os *charters* dizem aos testadores o que deve ser levado com consideração durante a realização dos testes (i.e., as diretrizes de teste), sem, no entanto, informar como eles devem executar os testes. Os charters são o principal artefato utilizado na empresa para a realização dos testes exploratórios. As informações contidas nos feedbacks relevantes entram como diretrizes extras para potencializar os testes, em termos de detecção de defeitos. Dessa forma, ao executar os testes exploratórios, os testadores levam em conta as diretrizes de ambas fontes. Esse módulo foi criado obedecendo esses critérios. Vejamos a Figura 11, que mostra como o protótipo disponibiliza os charters para a seleção.

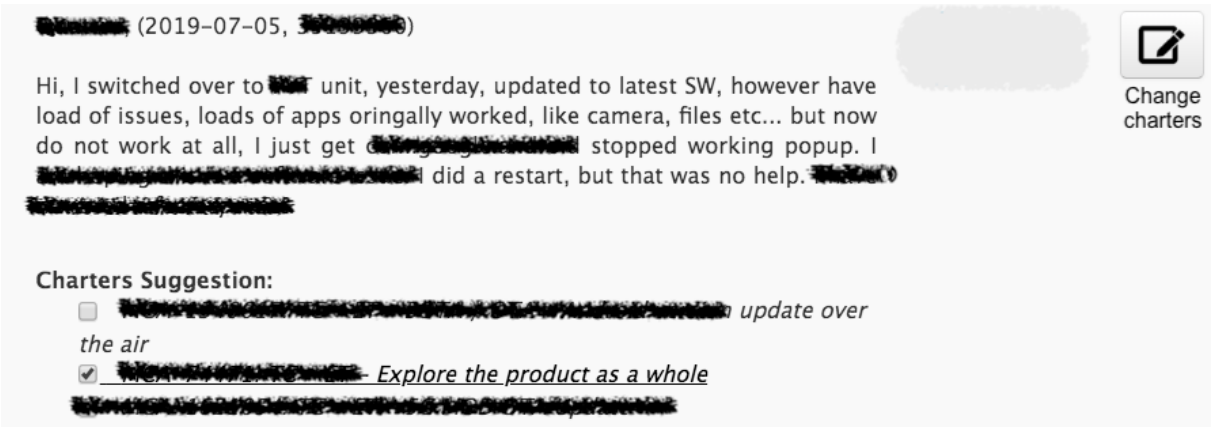

Figura 11: DAs – Busca e seleção de charters
Fonte: Autor (2018)

Nessa figura, os charters estão abaixo dos feedbacks, com o título *"Charters Suggestion"*, para dar uma conotação de que o protótipo está "sugerindo" charters ao feedback em questão. O testador, então, irá selecionar os charters que julgar mais relacionados aos feedbacks (via *check box*).

Tal como os feedbacks, os charters são armazenados em uma base de dados exclusiva, cuja diferença é que ela utiliza uma interface de consulta JSQL[11]. No entanto, essa interface é lenta e as buscas textuais são imprecisas. Para resolver esse problema, os charters são indexados mensalmente para uma outra estrutura de dados, que possui uma interface com um engenho de busca textual – *Apache Solr*. Essa ferramenta possibilita consultas textuais mais rápidas e precisas, além de permitir o uso de arquivos de configuração, como sinônimos e palavras irrelevantes, para tornar as consultas mais robustas. Tanto a indexação quanto a busca são feitos via Python, através da biblioteca *PySolr*[12].

No protótipo, internamente, os charters são recuperados do *Solr* através de alguns termos presentes nos feedbacks. De cada feedback, são identificados os termos que configuram *substantivos* (NN*, com uso da *part-of-speech tagging*[13], NLTK), os quais são utilizados como parâmetros de consulta para recuperação dos charters. Por exemplo, vejamos o seguinte feedback: *Video camera stopped respond when I receive a message.* Nesse feedback, podemos identificar como *substantivos* (NN*, em *part-of-speech*) os termos: *Video*, *camera* e *message*. Assim, esses 3 termos são utilizados como consulta no *Solr*, via *PySolr*.

Outro ponto importante é que, caso alguns charters não sejam recuperados para um dado feedback, o testador ainda pode busca-los através do botão *"Change charters"*. Vejamos a Figura 12 a seguir.

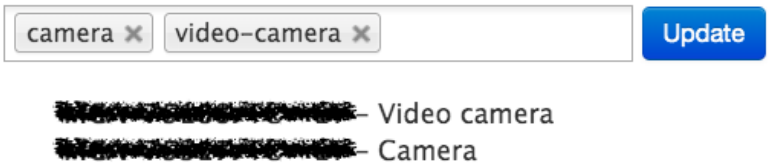

Figura 12: DAs – Mudança de charters
Fonte: Autor (2018)

[11] Consultas SQL codificadas em padrões JAVA. Consultar http://www.dba-oracle.com/t_jdbc_jsql.htm para maiores detalhes.
[12] Os charters no Solr são mantidos em *cores*, que são as estruturas utilizadas para indexar os dados, semelhante ao esquema de bancos de dados NoSQL. Através da biblioteca PySolr, as consultas são realizadas através do método **`search`**, invocado com o código: `Solr(core_link).search(consulta, n_linhas, campo_de_consulta)`. No protótipo, os substantivos identificados dos feedbacks são passados ao parâmetro `consulta`, para a recuperação dos charters. Os parâmetros `n_linhas` e `campo_de_consulta` são, respectivamente, a quantidade de charters a recuperar e o campo que deve ser utilizado para a consulta (i.e., o *sumário* do charter). O parâmetro `core_link` é o link do *core* criado via *Solr Admin* para a indexação e consulta dos dados. Para mais detalhes sobre o *Solr Admin*, verificar link https://lucene.apache.org/solr/guide/7_7/overview-of-the-solr-admin-ui.html.
[13] Para maiores detalhes, consultar link https://www.nltk.org/book/ch05.html sobre *part-of-speech tagging*, da biblioteca NLTK. Verificar também o trabalho de FERREIRA (2016).

Na figura, vemos duas *tags* e uma lista de charters relacionados a elas. Voltando um pouco na parte de *indexação*, quando os charters são recuperados da base JSQL para serem indexados no *Solr*, eles são transformados em objetos JSON[14], considerando 2 campos a priori: *summary* e *charter-key*, que são o título do charter e seu identificador único, respectivamente. No entanto, dado o problema da não recuperação de charters em alguns casos, um terceiro campo chamado *tag* é criado a partir do *summary*, que é identificador do charter criado a partir do resumo do campo *summary*. Assim, quando os charters vão ser indexados no *Solr*, eles ganham um terceiro campo, ficando: *summary, charter-key* e *tag*. Para criar as tags, foram utilizados métodos de *expressões regulares* (*regEx*) e remoção de *stopwords*. O objetivo é, através da tag, facilitar a identificação do charter quando o usuário precisar fazer uma busca por um charter apropriado a um dado feedback, caso ele não seja recuperado pelo protótipo. Por exemplo, para um charter com o título *Test all sound system*, a tag resultante seria "*sound-system*". O usuário pode adicionar e remover as tags no campo de texto. Uma tag pode ser usada para recuperar um ou mais charters. A seguir, basta acionar o botão "*Update*" para atualizar a lista de charters do feedback corrente. Vale ressaltar que o procedimento de indexação dos charters é feito através de um script à parte, que é executado 1 vez a cada mês.

4.2.2.4 Módulo de Apresentação e Submissão da análise de dogfooding

Este módulo consiste em permitir que o usuário submeta um documento textual ao sistema de gerenciamento de tarefas (Atlassian Jira), contendo uma lista dos charters selecionados e a que feedbacks eles estão associados. Quando, no módulo anterior, o testador seleciona um charter para um feedback, imediatamente aparece o botão *"Report Analysis"* no rodapé da tela do protótipo. Ao clicar nesse botão, a lista com os feedbacks e os charters relacionados aparecem. Essa lista é o documento da análise de dogfooding que deverá ser reportada. Vejamos a Figura 13 e 14, para maiores detalhes.

[14] JSON: JavaScript Object Notation. O Solr aceita vários outros formatos textuais para indexação dos dados, como *TXT* e *XML*.

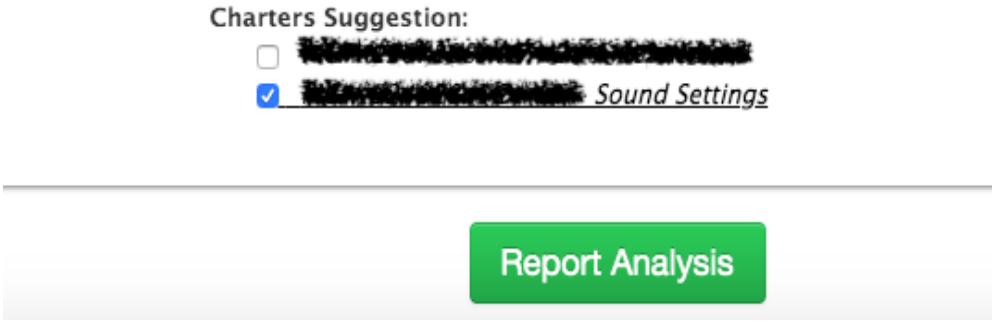

Figura 13: DAs – Reportar Análise de Dogfooding
Fonte: Autor (2018)

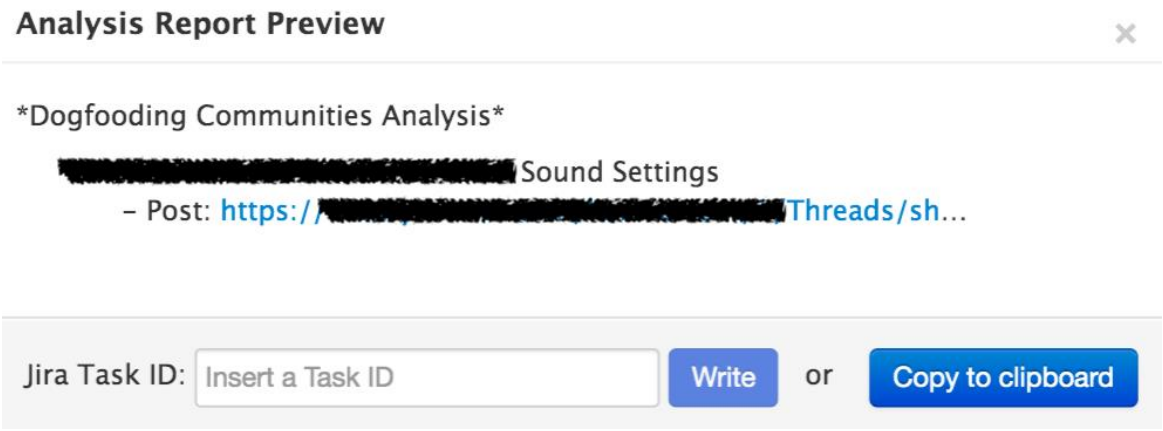

Figura 14: DAs – Documento da Análise de Dogfooding
Fonte: Autor (2018)

Na figura acima, vemos o *"Analysis Report Preview"*, onde estão o documento da análise (feedbacks e charters) e duas opções para realizar o report. O feedback é reportado com o link da comunidade, ao invés do texto. O testador pode inserir o ID da task do Jira relacionada ao aparelho e reportar nela diretamente (i.e., cada aparelho tem sua própria task de gerenciamento, onde estão as informações gerais das atividades de TEs) ou copiar o texto da análise e colar na mesma task. O formato do documento, bem como os dois mecanismos criados para realizar o report, foram criados de acordo com o que foi especificado pela equipe de TEs.

4.3 CONSIDERAÇÕES FINAIS

O protótipo foi desenvolvido com o intuito de automatizar boa parte dos procedimentos realizados manualmente pelos testadores durante as *análises de dogfooding*. O protótipo possui 4 funcionalidades principais, que consistem em (1) obtenção dos feedbacks a analisar; (2) classificação dos feedbacks; (3) busca e seleção dos charters relacionados; (4) apresentação e submissão dos resultados da análise. As demais funcionalidades são internas, como o *treinamento* do modelo de AM e *tratamento* dos dados.

A solução proposta teve como meta proporcionar um ganho de tempo e um ganho na identificação de alguns dos feedbacks relevantes que os testadores não conseguem recuperar com os procedimentos manuais. A solução partiu da utilização de algoritmos de *aprendizagem de máquina* para classificar os feedbacks, bem como de um sistema de RI (*Solr*) para a recuperação dos charters.

É importante lembrar que as funcionalidades levaram em conta o fluxo das atividades manuais realizados pelos testadores de TEs. Essa iniciativa foi útil para manter o padrão das atividades e facilitar a familiaridade com o protótipo. O próximo capítulo irá descrever, em detalhes, o desenvolvimento do protótipo e os testes realizados, principalmente no *treinamento*, na *classificação dos feedbacks* e na *recuperação de charters*, além dos testes de usabilidade.

5 IMPLEMENTAÇÃO E TESTES

O protótipo proposto foi desenvolvido para ser uma aplicação Web, fazendo uso de tecnologias como Python[15], Flask[16], AngularJS[17], JQuery[18] e as demais tecnologias padrão para construção de sistemas Web. Foram utilizadas duas bibliotecas *open-source* de *aprendizagem de máquina*, ambas para a linguagem Python: *Scikit-Learn*[19] e *Natural Language Toolkit*[20]. Também foi usada a plataforma *Solr*, com a biblioteca *PySolr*, para indexação e recuperação dos charters.

A seguir será apresentada a implementação das funcionalidades principais do protótipo, isto é, as funcionalidades individuais dos módulos citados no capítulo 4. Também serão apresentados os testes e resultados do *treinamento* e da *classificação de feedbacks*, bem como da *recuperação de charters*. Além disso, serão apresentados os *testes de usabilidade*, de modo a verificar o ganho de produtividade com o uso do protótipo em duas situações reais de análises de *dogfooding*.

5.1 COLETA/ARMAZENAMENTO DOS DADOS DE TREINAMENTO

Inicialmente, foram coletados dados para a construção do *corpus* de treinamento. Os dados das comunidades de *dogfooding* são periodicamente (i.e., a cada 1 hora) armazenados em tabelas do Google BigQuery[21] através de scripts criados por colaboradores de outros países. Esses colaboradores fazem uso desses dados em diversas aplicações e com diferentes objetivos.

Os dados de *dogfooding* foram transcritos para uma tabela exclusiva a ser utilizada pelo protótipo, que será apelidada de *dogfooding_feedbacks_base*. Essa tabela fornece tanto os dados de *treinamento* quanto os dados de *teste*. Os dados de

[15] Linguagem de programação orientada a objetos, funcional, interpretada e de tipagem dinâmica. Site: https://www.python.org/.
[16] Microframework escrito em Python para o desenvolvimento de aplicações Web simples. Site: http://flask.pocoo.org/.
[17] Framework JavaScript para o desenvolvimento de aplicações Web. Site: https://angularjs.org/.
[18] Biblioteca JavaScript para simplificação de scripts executáveis no browser. Site: https://jquery.com/.
[19] Biblioteca *open-source* de Machine Learning para Python. Para mais informações, ver http://scikit-learn.org/stable/.
[20] Conjunto *open-source* de bibliotecas de Processamento de Linguagem Natural (PLN) para Python. Mais informações, consultar link https://www.nltk.org/.
[21] Web service do *Google Cloud* utilizado para trabalhar com grandes quantidades de dados. Utiliza linguagem SQL. Para mais informações, ver o link https://cloud.google.com/bigquery/.

treinamento, particularmente, foram obtidos através da consulta SQL da Figura 15, a partir da interface de *query* do Google BigQuery:

```
SELECT content, creationdate
FROM [dogfooding_feedbacks_base]
WHERE creationdate
BETWEEN '2017-01-01' AND '2017-12-31'
```

Figura 15: Consulta para obtenção dos dados de treinamento
Fonte: Autor (2018)

A tabela *dogfooding_feedbacks_base* possui várias colunas (*content, source*, *country*, *author*, *creationdate*, etc.), porém apenas a coluna *content* é interessante para esta etapa, pois é ela que mantém o conteúdo textual de cada feedback. A coluna *creationdate* foi mantida para fins de atualização da base de treinamento, para substituição de feedbacks antigos por novos (ver seção 5.6, sobre *Retreinamento*).

Os dados de treinamento obtidos se referem a aparelhos do ano de 2017, considerando o uso do intervalo entre *janeiro* e *dezembro* na coluna *creationdate*. A consulta retornou 5.851 feedbacks. Desses, foram selecionados, de modo aleatório, 2.691 feedbacks, sendo salvos posteriormente num arquivo *.csv* para a realização da *etiquetagem manual*. Os feedbacks *relevantes* receberam a label **1** e os *irrelevantes* a label **0**, resultando em 1488 amostras *relevantes* e 1203 amostras *irrelevantes*.

Concluída a etiquetagem, as amostras foram submetidas aos algoritmos de *pré-processamento* para padronização dos dados. Finalmente, as amostras foram salvas em outro arquivo *.csv*, dessa vez com o formato *[texto processado, label]*. Os dados foram armazenados já pré-processados, ficando prontos para serem submetidos aos algoritmos de classificação. Para o *pré-processamento*, foram utilizados os métodos do script *text_normalize* (ver seção a seguir). Abaixo, a Figura 16 traz o modelo BPM que demonstra esse processo.

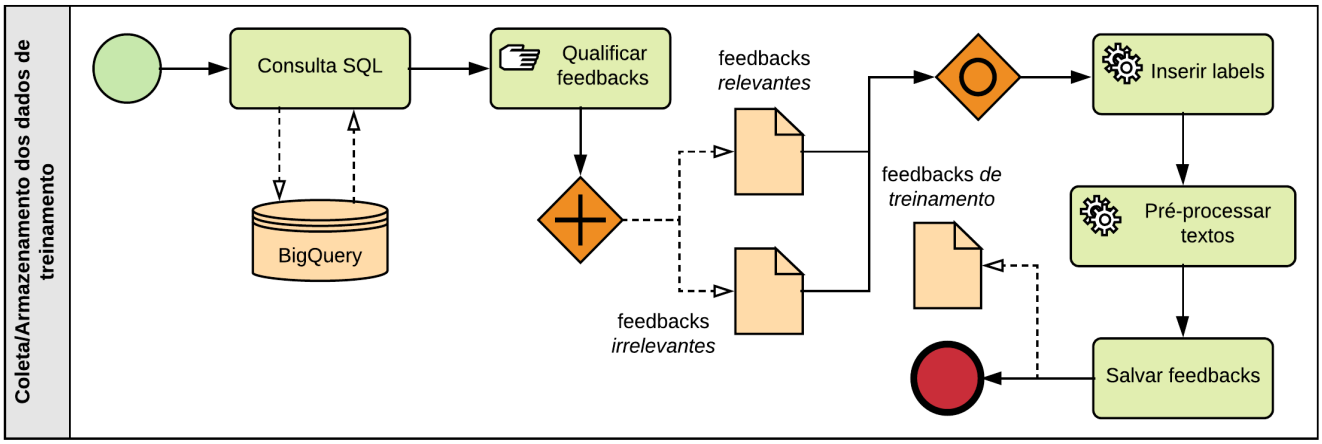

Figura 16: Diagrama BPMN para a Coleta e Armazenamento dos dados de treinamento
Fonte: Autor (2018)

5.2 PRÉ-PROCESSAMENTO

O pré-processamento dos textos é um procedimento essencial à aplicação dos algoritmos de AM, responsável por formatar os dados apropriadamente e remover anomalias, redundâncias e informações irrelevantes, que contribuem fortemente em erros nos resultados produzidos pelos algoritmos. Assim, foi implementado o script *text_normalize*, que provê os métodos para o tratamento e a normalização dos textos, para serem aplicados nos dados de treinamento e nos dados de teste.

Os métodos foram criados com base nas técnicas mencionadas na seção 3.2.2: (1) *tokenize*: particionar termo a termo o texto de cada amostra do corpus textual; (2) *remove_stopwords*: remove do texto as *stopwords* presentes nele; (3) *remove_chars*: responsável pela remoção de caracteres especiais, dígitos e pontuações; (4) *stemming_text*: responsável por reduzir uma palavra ao seu radical. Aqui, foi utilizado algoritmo *Potter*; (5) *tfidf_vectorizer*: responsável pela criação da representação vetorial dos documentos, com base na medida TF-IDF; (6) *feature_selector*: responsável por criar uma representação reduzida dos dados, mantendo as principais características dos documentos.

Vale salientar que esses métodos não são aplicados aleatoriamente, mas obedecem uma certa ordem para garantir que os dados sejam corretamente normalizados. Por isso, todos eles são chamados dentro de um método principal, nomeado de *normalizer*. Esse método realiza as chamadas segundo essa ordem: *tokenize* → *remove_chars* → *remove_stopwords* → *stemming_text* → *tfidf_vectorizer* → *feature_selector*. Vejamos essa ordem expressa no diagrama BPMN da Figura 17.

É importante ressaltar o método *feature_selector* foi testado na fase de *Treinamento* antes de ser colocado em produção.

O diagrama a seguir mostra o fluxo do *pré-processamento* assumindo que todos esses os procedimentos já foram testados. Outro ponto é que o uso do método *feature_selector* em produção é opcional, pois depende da avaliação do seu uso no *Treinamento*.

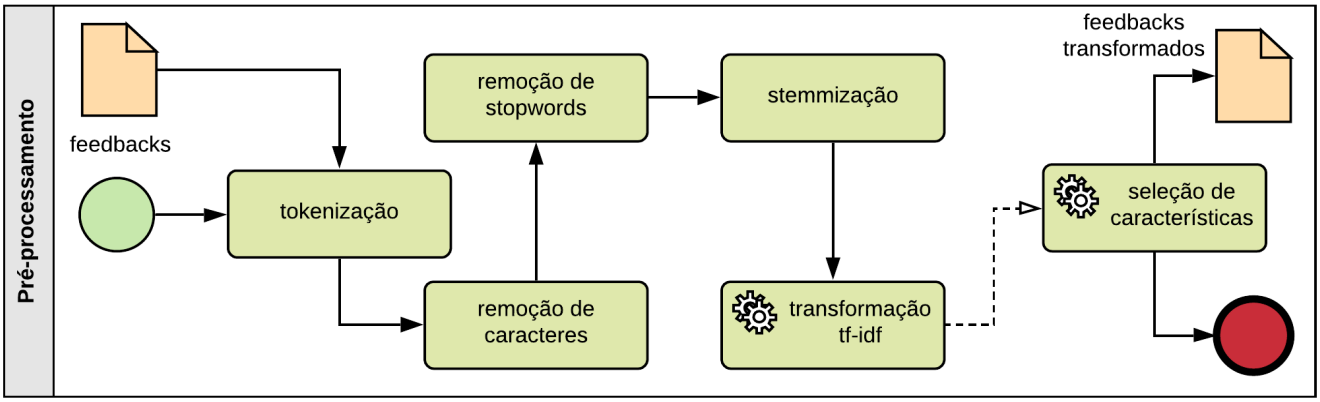

Figura 17: Diagrama BPMN do procedimento de Processamento
Fonte: Autor (2018)

Para exemplificar, tomemos uma amostra de um feedback antes e depois do *pré-processamento* (não considerando os métodos de representação):

> Feedback original: *The auto flashlight stopped work, on current version 25.231. Only work if I activate myself.*
>
> Feedback normalizado: *auto flashlight stop work current version work activ manual*

5.3 CRIAÇÃO DOS CLASSIFICADORES - FASE DE TREINAMENTO

Como já mencionado, o *treinamento* é o processo que cria os modelos de *aprendizagem de máquina* a partir de um conjunto etiquetado de amostras de dados. Os dados de treinamento foram utilizados para construir 3 modelos de classificação de diferentes abordagens. Os 3 modelos foram selecionados de acordo com os algoritmos citados na seção 3.4.2, os quais são: Naïve Bayes, K-Nearest Neighbors e Support Vector Machines.

Os procedimentos de treinamento foram realizados por meio do script *train_classifier*, que consiste em obter as amostras de treinamento processadas e transformadas; treinar o modelo com as amostras; e persistir o modelo em arquivo.

Vejamos a seguir o passo a passo da execução desse script:

(1) Carrega os dados de treinamento, isto é, os feedbacks de treinamento pré-processados (pelo script *text_normalize*, na seção 4.3.2);

(2) Executa o treinamento do modelo de classificação selecionado, dentre os 3 modelos abaixo:

(a) *Multinomial Naïve Bayes* (MNB): variante do NB convencional, utilizado em tarefas de *classificação textual* em que os dados são distribuídos de forma *multinomial* (*vetores de contagem de termos*). A *distribuição multinomial* é representada por vetores no formato $\theta_y = (\theta_y 1, \ldots, \theta_y n)$ para cada classe *y*, em que *n* é o número de termos e θ_{yi} é a probabilidade $P(x_i \mid y)$ do termo *i* estar presente em uma amostra pertencente à classe *y* (SCIKIT-LEARN, 2018);

(b) *C-Support Vector Classification* com kernel *linear*: *C* é um valor numérico que serve como um *otimizador*, ajustando a *penalidade de erro* para *falsos positivos*/*negativos* e *outliers*[22]. O kernel *linear* foi escolhido por ser mais adequado à *classificação textual*. Isso porque a maioria dos problemas de classificação de textos linearmente separáveis, os textos possuem muitos atributos (palavras), são menos parâmetros para otimizar (apenas o parâmetro *C*) e o treinamento é mais rápido do que com outros kernels (KOWALCZYK, 2014);

(c) *K-Neighbors Classifier*: O objetivo é testar o KNN com diferentes métricas e valores de *k*, a fim de verificar a configuração que provê melhores resultados de acurácia.

(3) O modelo criado é persistido para ser utilizado no procedimento de *classificação*.

Vejamos, na Figura 18, um resumo desse processo num diagrama BPMN.

[22] Valores atípicos e/ou inconsistentes em relação aos demais valores de uma população.

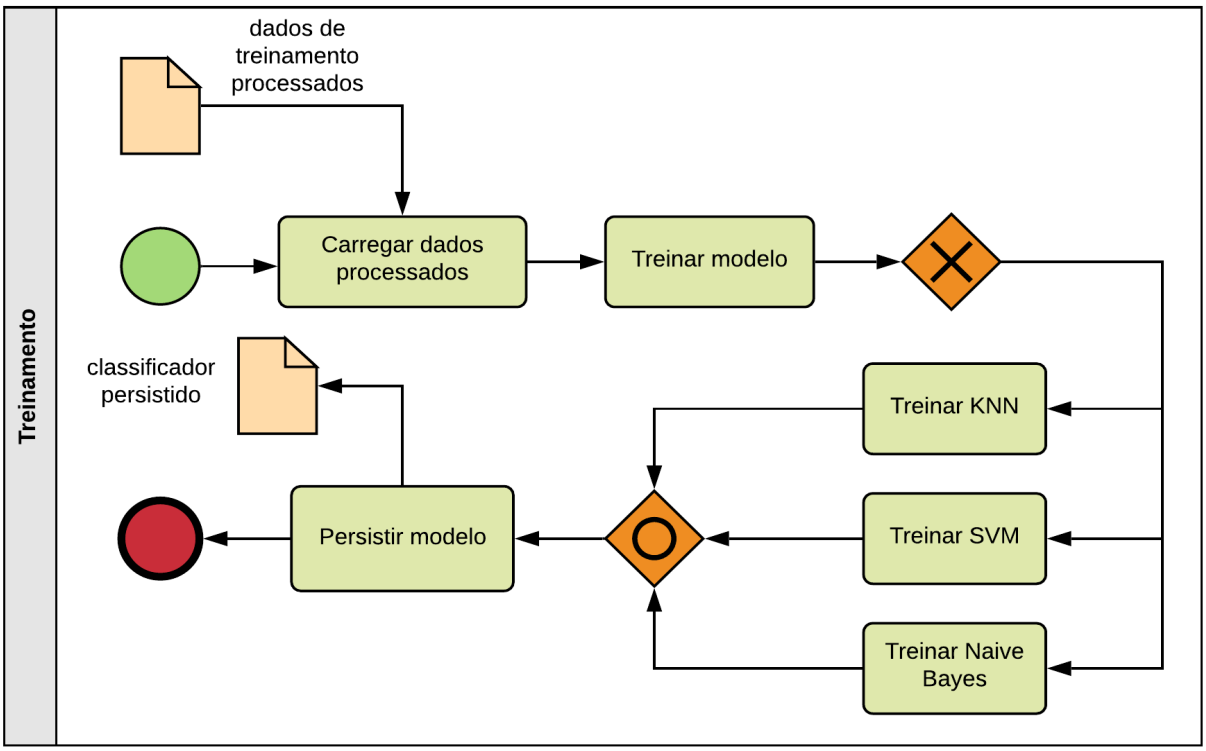

Figura 18: Diagrama BPMN do procedimento de Treinamento
Fonte: Autor (2018)

5.3.1 Treinamento dos Modelos – Metodologia

O objetivo dessa fase consiste na criação e avaliação dos modelos, com base nos dados de *treinamento*. Para isso, foram utilizados métodos para avaliar a capacidade de generalização dos modelos, para tentar ampliar essa capacidade e também para otimizar o processo de treinamento em termos de uso dos atributos mais representativos. Vejamos cada um dos métodos utilizados.

(1) *Seleção de Atributos*: como visto na seção 3.2.2, também é um método de *pré-processamento*, cujo objetivo é selecionar um menor conjunto de atributos mais representativo dos dados, para reduzir a dimensionalidade. Aqui, foi utilizado o método de *Seleção Univariada*[23], com o uso da classe *SelectKBest*, do pacote *sklearn.feature_selection*, usando as métricas *Informação Mútua* e *X-quadrático*.

[23] Método que utiliza testes estatísticos para selecionar os atributos que são mais determinantes para uma dada variável-alvo.

(2) *Validação Cruzada*: como mencionado na seção 3.4.1, esse procedimento tem como objetivo avaliar se um modelo de classificação é eficiente o bastante para classificar novas instâncias de dados (instâncias não vistas pelos modelos no treinamento), através do particionamento iterativo dos dados de *treinamento* (K-folds).

(3) *Busca de hiperparâmetros*: método que realiza buscas exaustivas de hiperparâmetros para os modelos. Consiste em encontrar o modelo ótimo, com a melhor configuração de hiperparâmetros, dada uma lista de valores para os mesmos. Realiza internamente uma *Validação Cruzada* para cada uma das configurações, a fim de avaliar qual a melhor delas. Foi utilizado de acordo com a abordagem de THAKUR (2016).

Sabendo quais métodos foram utilizados, vejamos o passo a passo dessa fase. Inicialmente, os dados precisaram ser carregados para uma estrutura de fácil manipulação. Para isso, foi utilizada a biblioteca *pandas*, que fornece estruturas robustas para manipular dados. Conforme a seção 5.1, o corpus possui 2691 amostras, sendo 1488 de feedbacks *relevantes* e 1203 de feedbacks *irrelevantes*.

Com os dados carregados, o próximo passo foi criar a representação vetorial, utilizando a classe *TfidfVectorizer*, do pacote *sklearn.feature_extratcion.text*. A saída da transformação TF-IDF foi uma *matriz esparsa*[24] com 5094 atributos, ou seja, 5094 palavras do vocabulário do corpus.

Em seguida, essa matriz resultante foi particionada aleatoriamente em dois conjuntos de dados: *treino*, com 70% dos dados e *teste* com 30% dos dados. O particionamento ocorreu de modo *estratificado*, para garantir a mesma distribuição de dados para ambos conjuntos. Os dados de *treino* foram utilizados na *busca exaustiva* para encontrar os melhores parâmetros para os modelos, com o uso da classe *GridSearchCV* do pacote *sklearn.model_selection*.

A *seleção de atributos*, com o método de *Seleção Univariada,* também foi adicionada à *Grid Search*. O parâmetro de *cv*[25] foi K-folds (estratificado) igual a 10, com métrica *F1*. Especificamente, o Naïve Bayes foi utilizado com a *Grid Search*

[24] Matriz com grande quantidade de elementos nulos (iguais a 0). Como dito antes, o TF-IDF cria uma representação em que cada amostra corresponde a um vetor do tamanho do vocabulário criado (5094 atributos), e cada palavra da amostra existente no vocabulário recebe um peso que indica a importância do termo para a amostra. As palavras não existentes recebem 0, que são a maioria.
[25] *Cv*: parâmetro de validação cruzada da *GridSearchCV*.

somente com a *seleção de atributos*, pois ele não necessita de configuração de hiperparâmetros, de acordo com a abordagem de THAKUR (2016). Com exceção dessa configuração, o Naïve Bayes foi avaliado utilizando a *Validação Cruzada* convencional, também com o parâmetro K-folds igual a 10 e métrica *F1*. Os hiperparâmetros utilizados para os outros modelos e *seleção de atributos* foram: (1) KNN: *metric* {*cosine*, *euclidean*}; *neighbors* {*5, 13, 27, 33, 49*}; SVM: *C* {*0.001, 0.01, 0.1, 1, 10, 100*}; Seleção de Atributos: *k* atributos {*1000, 1500, 2000*}. Depois disso, o próximo passo foi avaliar os modelos utilizando duas configurações em relação aos dados: (1) *original* e (2) *seleção de atributos*. Os procedimentos de avaliação foram realizados através da aplicação *Jupyter Notebook*[26]. Vejamos os resultados a seguir.

5.3.2 Resultados

1. Original: dados sem qualquer transformação além do TF-IDF. A Figura 19 e o Quadro 2, a seguir, apresentam os resultados:

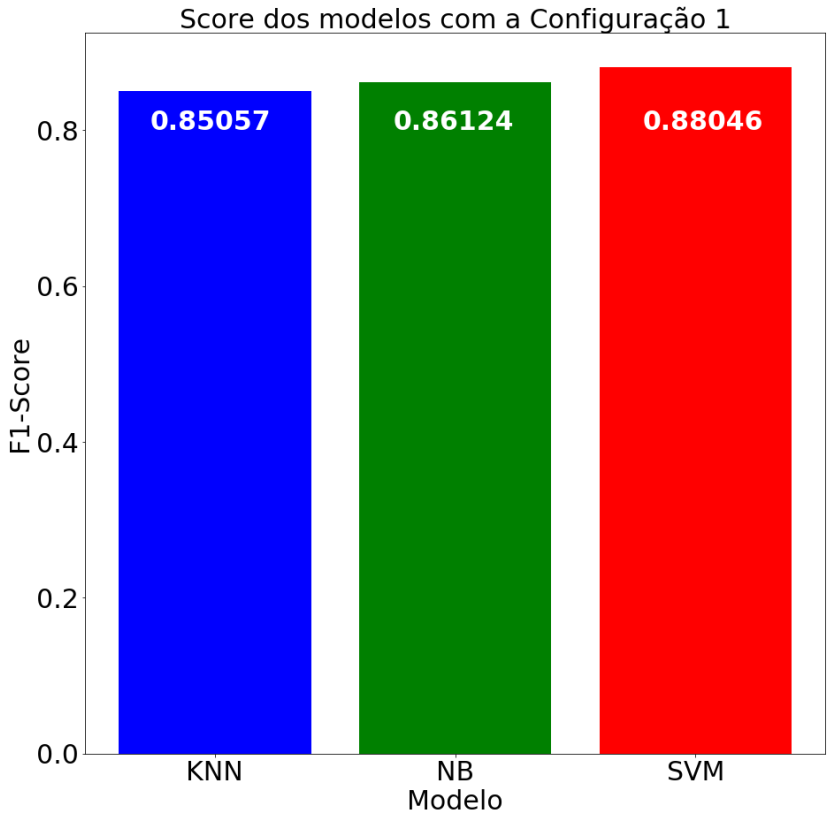

Figura 19: F1-Score dos modelos com a Configuração 1
Fonte: Autor (2018)

[26] Site oficial do Jupyter Notebook: https://jupyter.org/.

Classificador	melhores parâmetros
Naïve Bayes	N/A
KNN	vizinhos=**49**; métrica=**cosseno**
SVM	C=**1**

Quadro 2: Melhores parâmetros da Configuração 1
Fonte: Autor (2018)

2. Seleção de atributos: aplicação do método de *seleção de atributos* nos dados, com as métricas Informação Mútua e X-quadrático. Vejamos os resultados na Figura 20 e no Quadro 3.

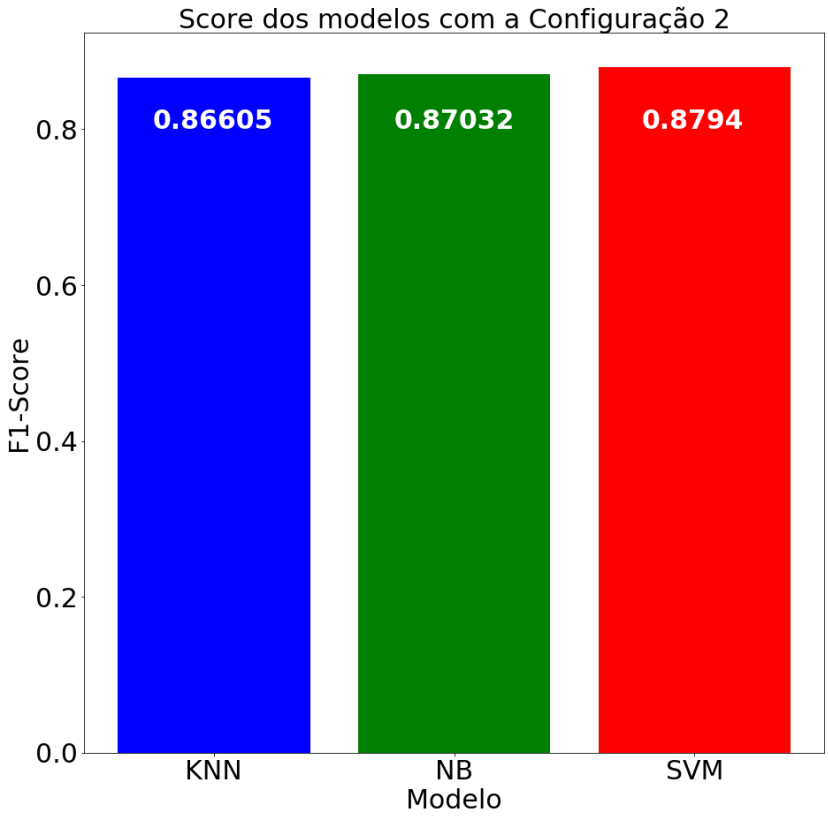

Figura 20: F1-Score dos modelos com a Configuração 2
Fonte: Autor (2018)

Classificador	**melhores parâmetros**
Naïve Bayes	redução=**1500**; métrica de redução=**mutual_info**
KNN	redução=**1000**; métrica de redução=**x^2**; vizinhos=**13**; métrica=**cosseno**
SVM	redução=**2000**; métrica de redução=**x^2**; C=**1**

Quadro 3: Melhores parâmetros da Configuração 2
Fonte: Autor (2018)

Observando-se os resultados acima, percebe-se que todas as configurações obtiveram resultados muito próximos e satisfatórios. Na configuração 1, por exemplo, o melhor modelo foi o SVM, com 88.05% de F1-score, seguido pelo Naïve Bayes, com 86.12%, e pelo KNN, com 85.05%. Os melhores parâmetros dos modelos foram: KNN {*vizinhos*: 49, *métrica*: cosseno}; SVM: {*C*: 1}. O tempo gasto no treinamento para essa configuração, com os 3 modelos, foi de: 74 milissegundos para o NB; 5.69 segundos para o KNN; 22.7 segundos para o SVM. Na configuração 2, os scores do Naïve Bayes e do KNN tiveram um aumento de 1.55% e 0.9%, resultando em 86.60% e 87.03% respectivamente em relação à configuração 1. O SVM teve uma diminuição irrisória de 0.1%, resultando em 87.94%. Os melhores parâmetros dos modelos foram: Naïve Bayes {*redução*: 1500, *métrica de redução*: informação mútua}; KNN {*vizinhos*: 13, *métrica*: *cosseno*, *redução*: 1000, *métrica de redução*: x-quadrático}; SVM {*C*: 1, *redução*: 2000, *métrica de redução*: x-quadrático}. O tempo gasto no treinamento para essa configuração, com os 3 modelos, foi de: 1 minuto e 44 segundos para o NB; 17 minutos para o KNN; 11 minutos e 17 segundos para o SVM.

Apesar de um maior gasto de tempo, o treinamento com a *seleção de atributos* foi favorável a todos os modelos. O KNN, por exemplo, além do pequeno aumento da F1-score, reduziu o número de atributos para apenas 1000, contribuindo também para a diminuição do número de *vizinhos*, com diferença de 36 vizinhos em relação à configuração 1. O Naïve Bayes beneficiou-se da mesma forma, com o aumento de score e uma redução para 1500 atributos. O SVM, apesar de seu score ter sido praticamente mantido, teve uma redução de atributos para 2000.

Para os modelos KNN e SVM, em especial, essa redução de atributos foi importante pelo fato de eles exigirem alto custo computacional, principalmente para o procedimento de treinamento. Com a redução, esse custo tende a diminuir bastante.

Outro ponto importante é que, apesar da aparente perda de informação, comparando-se as configurações, os resultados demonstram que os atributos mais informativos de cada classe foram mantidos. Nota-se, ainda, que a configuração 2 sobressaiu-se em relação à configuração 1. Dessa forma, utilizaremos os modelos da configuração 2 para a realização dos *testes*, com os 30% dos dados reservados para tal. A partir desses testes, poderemos verificar qual modelo demonstra ser o mais eficiente para ser utilizado no protótipo.

5.4 CLASSIFICAÇÃO DE FEEDBACKS

Essa etapa é bem mais simples do que a anterior, pois todos os procedimentos já foram realizados para a criação do modelo de classificação. Basta apenas utilizá-lo para categorizar as novas instâncias. Para isso, foi criado um script denominado *classifier*. Esse script consiste basicamente em carregar o modelo de classificação, obter os novos feedbacks de dogfooding de um aparelho e classificá-los e retornar os relevantes para o usuário. Vejamos a seguir o passo a passo desse script:

(1) Obter novos feedbacks e normalizá-los. Esses feedbacks são recuperados via script (*data_collector*), que possui uma consulta SQL interna semelhante à da Figura 21:

```
SELECT
     device, content, url, creationdate
FROM [dogfooding_feedbacks_base]
WHERE creationdate BETWEEN previous_date AND current_date
AND device = device_name
ORDER BY creationdate DESC LIMIT number_feedbacks
```

Figura 21: Exemplo de consulta para obtenção dos dados de teste
Fonte: Autor (2018)

Essa consulta recupera dos feedbacks: o *nome do aparelho* (device), o *conteúdo textual* (*content*) deles, *url de acesso* às comunidades onde os feedbacks são reportados e a *data de criação* (*creation_date*) deles. Os feedbacks são recuperados do mais recente ao mais antigo. Em seguida, os conteúdos textuais (coluna *content*) dos feedbacks passam pelo *pré-processamento* (*text_normalizer*) para padronizá-los, de modo a deixá-los na mesma estrutura dos dados de treinamento;

(2) Carregar modelo de classificação. Aqui, subentende-se que o modelo em questão já está treinado e persistido;

(3) Os novos feedbacks normalizados são submetidos ao classificador para serem classificados como *relevantes* ou *irrelevantes*;

(4) Os feedbacks classificados como relevantes são entregues ao usuário via interface gráfica (GUI).

A Figura 22, a seguir, resume esse processo num diagrama BPMN:

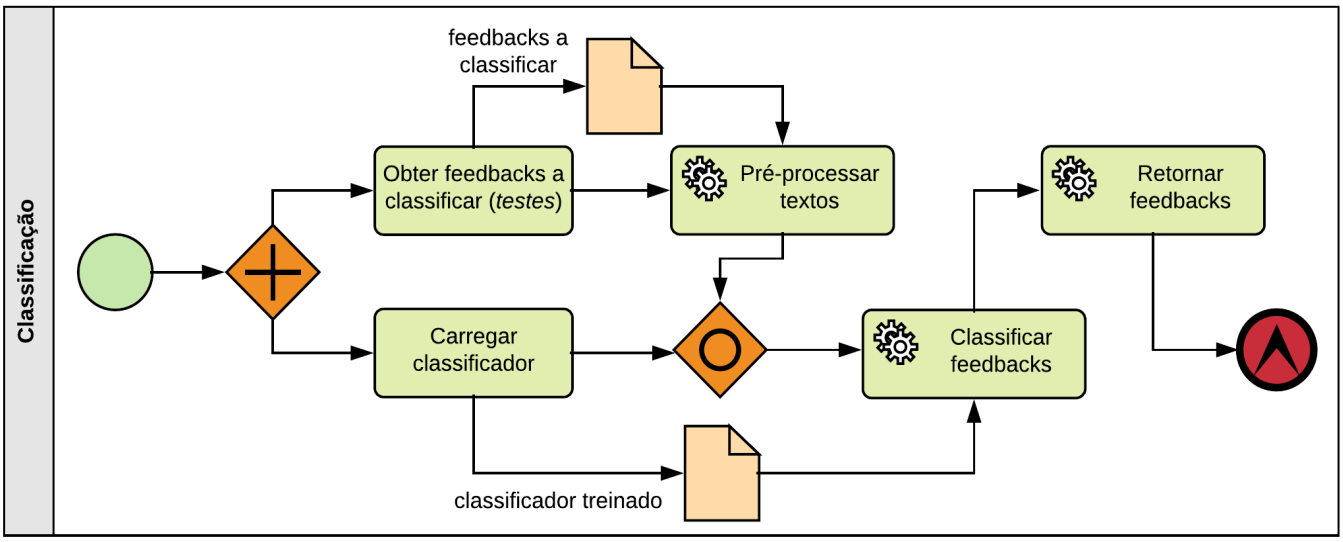

Figura 22: Diagrama BPMN do procedimento de Classificação
Fonte: Autor (2018)

5.4.1 Classificação – Testes e Resultados

Como dito na seção 5.3.1, o próximo passo foi testar os modelos da configuração 2 utilizando a base reservada para testes, que dispõe de 30% dos dados (808 amostras) não vistos pelos modelos durante o treinamento. Foram utilizadas as métricas de *Precisão*, *Cobertura*, *F1-score* e *Acurácia* de cada modelo. Utilizamos também a *Matriz de Confusão* para demonstrar a frequência da classificação de cada variável-alvo. Vejamos os resultados a seguir.

5.4.1.1 Naïve Bayes

acurácia	0.858911
f1-score	0.879747
precisão	0.822485
cobertura	0.945578

Tabela 2: Resultados da classificação do Naïve Bayes
Fonte: Autor (2019)

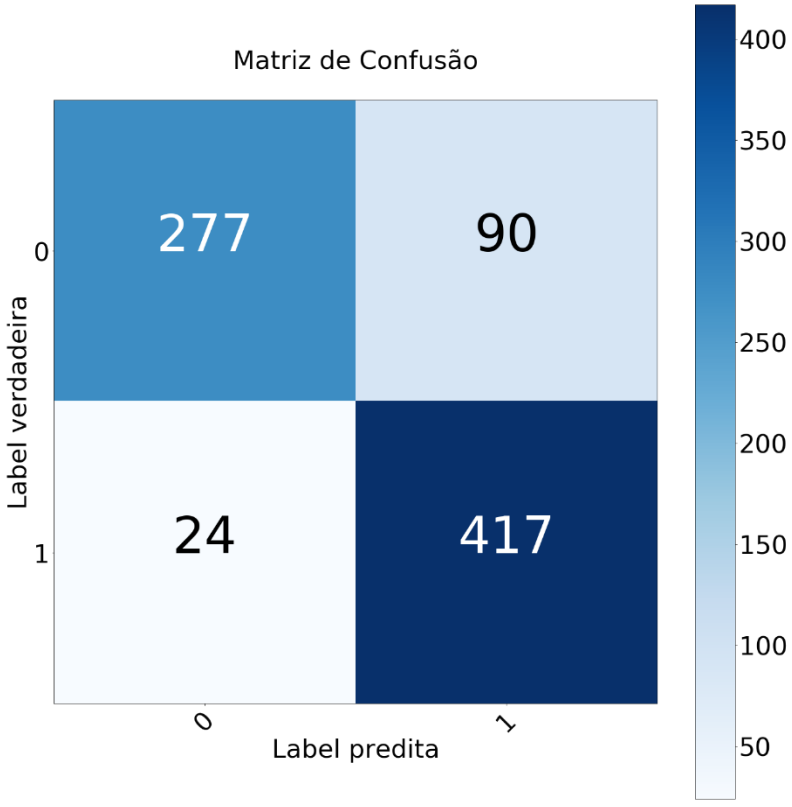

Figura 23: Matriz de confusão do Naïve Bayes
Fonte: Autor (2019)

5.4.1.2 KNN

acurácia	0.841584
f1-score	0.863248
precisão	0.816162
cobertura	0.916100

Tabela 3: Resultados da classificação do KNN
Fonte: Autor (2019)

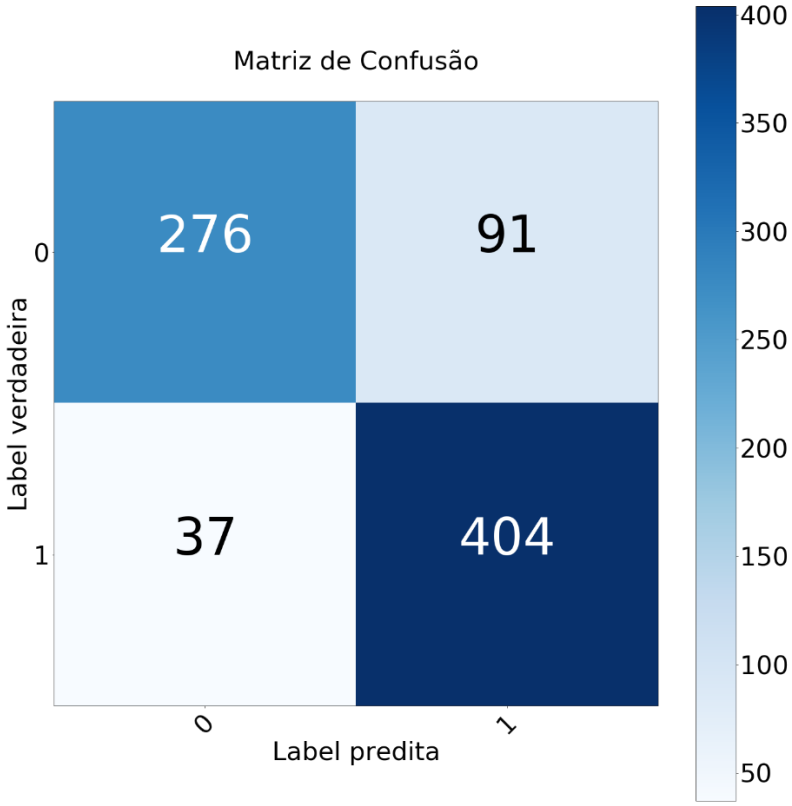

Figura 24: Matriz de confusão do KNN
Fonte: Autor (2019)

5.4.1.3 SVM

acurácia	0.878713
f1-score	0.885514
precisão	0.913253
cobertura	0.859410

Tabela 4: Resultados da classificação do SVM
Fonte: Autor (2019)

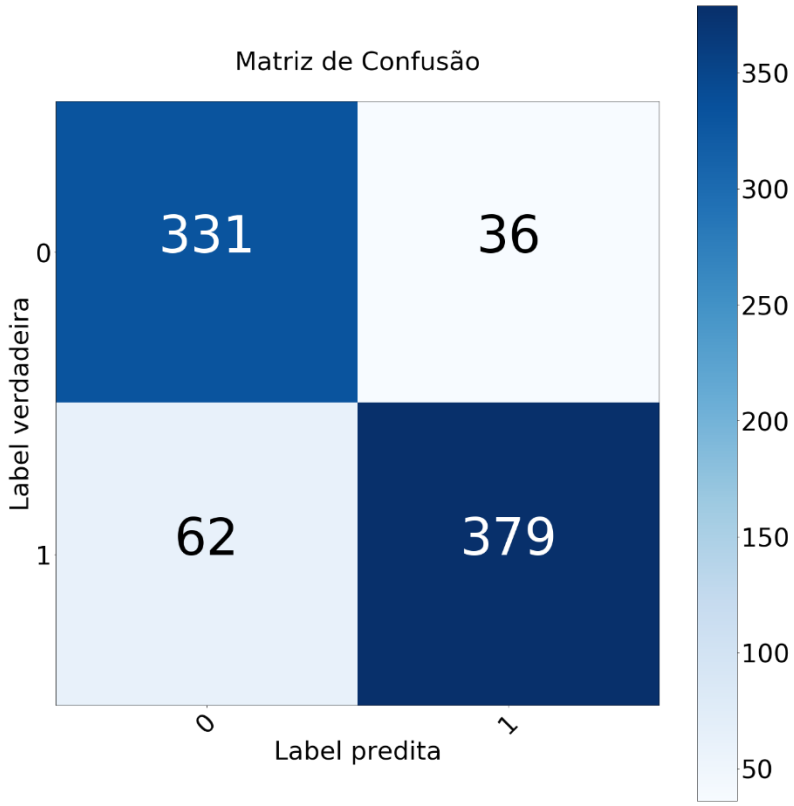

Figura 25: Matriz de confusão do SVM
Fonte: Autor (2019)

Dados os resultados acima, é possível ordenar os modelos quanto à sua eficiência, em termos gerais, da seguinte forma: 1º SVM, 2º Naïve Bayes e 3º KNN. O Naïve Bayes teve um ganho aproximado de 2% em relação ao KNN nas métricas de *precisão*, *acurácia* e *f1-score*. Na *cobertura*, houve um ganho de 3% também para o Naïve Bayes. Se observarmos, no entanto, as *matrizes de confusão* desses dois modelos, a diferença da distribuição das amostras é irrisória. Por exemplo, houve 277

verdadeiros negativos do NB contra 276 do KNN. Nos *verdadeiros positivos*, a diferença foi de apenas 13 amostras a mais para o NB. Dessa forma, mesmo o NB tendo obtido um ganho maior nos resultados das métricas, pode-se concluir que os resultados do NB e o KNN são razoavelmente equivalentes.

O SVM se saiu um pouco melhor que os demais, principalmente nas métricas de *precisão* e *acurácia*. Em termos de *precisão*, o SVM obteve um ganho de aproximadamente 10% sobre os outros modelos. Esse ganho de *precisão* é demonstrado nas *matrizes de confusão*, em que o NB e o KNN tiveram 90 e 91 exemplos de *falsos positivos* respectivamente, contra apenas 36 do SVM. Em relação à *acurácia*, obteve ganho de 2% sobre o NB e de 3.7% sobre o KNN. Em relação à *F1-score*, o ganho significativo ocorreu apenas sobre o KNN, com 2.2%. A *cobertura*, no entanto, foi mais baixa do que a obtida pelos outros modelos. Em comparação com o NB, caiu 8.6%; e em comparação com o KNN, caiu 5.6%. Essa queda deve-se à quantidade de *falsos negativos* do SVM, sendo 62 amostras contra 24 do NB e 37 do KNN.

Frente a essas análises, vê-se que a qualidade do SVM, no geral, foi a melhor, garantindo maior percentual nas 3 primeiras métricas. Apesar da diminuição na *cobertura*, com um maior número de *falsos negativos*, o SVM conseguiu garantir uma distribuição balanceada para ambas classes, isto é, das amostras corretamente classificadas. Com base na *matriz de confusão* do SVM, isso fica bem evidente, visto que foram 331 amostras de *verdadeiros negativos* e 379 amostras para os *verdadeiros positivos*.

Apesar das conclusões acima, tanto em relação à equivalência razoável entre os modelos NB e KNN quanto em relação ao SVM como melhor modelo, *testes estatísticos* se fazem necessários para que tenhamos uma certeza estatística dessas conclusões. Dessa forma, a seção a seguir irá tratar dos *testes estatísticos* realizados com os três modelos de aprendizagem.

5.4.2 Testes Estatísticos e Resultados

Os *testes estatísticos* foram utilizados para comparar a eficiência dos modelos, com o intuito de saber, estatisticamente, qual o melhor modelo que deve ser utilizado para a tarefa de *classificação de feedbacks*. Os procedimentos de teste realizados foram:

(1) *Validação*: consiste em executar *N* vezes o processo de *treinamento* e *classificação*. Em outras palavras, são realizados *N* vezes: o particionamento dos dados (*train* e *test*); a vetorização TF-IDF; a GridSearch; a classificação dos dados de teste; e o armazenamento das pontuações de *acurácia* e *F1-score*. Esse procedimento foi realizado para cada um dos 3 modelos. O objetivo deste procedimento é verificar a consistência dos modelos, observando a variabilidade dos resultados das métricas.

(2) *Teste de normalidade*: testa a *hipótese nula* de que a população tem uma distribuição normal. Foi aplicado às métricas dos modelos utilizando *Shapiro-Wilk*[27].

(3) *T-teste*[28]: teste paramétrico que verifica a *hipótese nula* de que duas populações independentes possuem valores médios equivalentes. Neste teste, há a suposição de que as populações têm variância idêntica por padrão. Também é necessário que o *valor-p* resultante do *teste de normalidade* seja maior que 5%. Através desse teste, é possível saber se dois modelos de AM são estatisticamente equivalentes.

Primeiramente foi realizado o procedimento de *Validação*, com valor de *N* igual a 30 (i.e., 30 execuções). Os resultados de média (*mean*) e desvio-padrão (*std*) das métricas estão na Tabela 5:

	mean (f1)	std (f1)	mean (acc)	std (acc)
Naïve Bayes	0.875557	0.009135	0.853119	0.011276
KNN	0.861101	0.009597	0.836638	0.012833
SVM	0.879640	0.012645	0.870962	0.012175

Tabela 5: média e desvio-padrão das métricas após a Validação
Fonte: Autor (2019)

[27] Consultar *Shapiro-Wilk* na biblioteca *SciPy.org* para maiores detalhes: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html#id3
[28] Consultar *T-test* na biblioteca *Scipy.org* para maiores detalhes: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html

Observando a Tabela 5, nota-se que os modelos se mantiveram consistentes. Na métrica *f1* (F1-score), o NB e o KNN tiveram variações razoavelmente iguais (desvio-padrão), com diferença de pouco menos de 2% na média. O SVM teve maior média, com acréscimo de menos de 1% para o NB e 2% para o KNN. O SVM também teve maior desvio-padrão, isto é, teve uma variabilidade de f1 levemente maior que os demais modelos. Já na métrica *acc* (Acurácia), os valores de desvio-padrão mostram que as variações entre os modelos foram bem equilibradas. Já em termos de média, o SVM permanece na frente, com ganho de 1.7% e 3.4% em relação ao NB e KNN respectivamente.

Vejamos agora os resultados do teste de *Shapiro-Wilk*, para verificação da normalidade dos dados (Tabela 6).

	valor-p (f1)	valor-p (acc)
Naïve Bayes	0.079514	0.091821
KNN	0.085720	0.721075
SVM	0.212603	0.126467

Tabela 6: *valor-p* do teste de Shapiro-Wilk para as duas métricas
Fonte: Autor (2019)

Com base nos resultados da Tabela 6, observa-se que as métricas que todas as métricas têm distribuição normal, logo a hipótese nula não é rejeitada (*valor-p* obtido do teste é maior que 5%).

Verificada a normalidade dos dados, podemos passar para o *T-teste*. Aqui, comparamos as métricas dos modelos 2 a 2. Do mesmo modo que o teste *Shapiro-Wilk*, também verificamos se o *valor-p* resultante é maior que 5%. Vejamos os resultados na Tabela 7 abaixo.

	NB x **KNN**	**NB** x **SVM**	**KNN** x **SVM**
valor-p (f1)	$8.74 * 10^{-6}$	0.155904	$5.24 * 10^{-6}$
valor-p (acc)	$3.431 * 10^{-5}$	$1.06 * 10^{-7}$	$4.14 * 10^{-12}$

Tabela 7: *valor-p* do T-teste para as duas métricas
Fonte: Autor (2019)

De acordo com a Tabela 7, todos modelos são estatisticamente diferentes, exceto no caso do valor-p do Naive Bayes versus SVM da métrica f1 (em vermelho), que não rejeita a hipótese nula. Podemos dizer que, em termos de F1-score, esses dois modelos são estatisticamente equivalentes. Em contrapartida, na métrica de acurácia, são estatisticamente diferentes, isto é, o SVM se sobressai em relação ao NB.

Levando em conta os resultados já observados na classificação e na validação, o SVM ainda permanece em vantagem. Apesar do NB ter tido um bom desempenho e de utilizar um baixo custo computacional, o SVM ainda o supera em termos de *generalização*, que é um fator importante para o problema de classificação deste trabalho especificamente. Dessa forma, o SVM foi escolhido para ser utilizado no protótipo do sistema.

Todos os testes que foram realizados também demonstraram que as técnicas de *aprendizagem de máquina* podem ser aplicadas, de maneira eficiente, à tarefa de *classificação de feedbacks*, podendo assim substituir o procedimento manual.

5.5 RECUPERAÇÃO DE CHARTERS E SUBMISSÃO DA ANÁLISE DE DF

Esta funcionalidade é responsável pela *recuperação de charters* de TEs para os feedbacks relevantes bem como pela realização da *submissão da análise de DF*. Como dito antes (seção 4.2.2.3), o protótipo recupera os charters relacionados aos feedbacks a partir de alguns termos desses feedbacks (i.e., substantivos). Os charters são indexados periodicamente para o *Solr* (via script à parte) e a recuperação é realizada internamente com o uso da biblioteca *PySolr*, no momento em que a busca pelos feedbacks a serem classificados é realizada. O protótipo apresenta, para cada feedback relevante, uma lista dos possíveis charters relacionados, e o usuário pode selecionar 1 ou mais charters. Após os charters são selecionados, o usuário realiza a submissão da análise (i.e., os charters selecionados/feedbacks relevantes).

Internamente, esse processo provê: indexação periódica dos charters através do *Solr*; busca dos charters utilizando os termos dos feedbacks (substantivos) como consulta; apresentação dos charters retornados para cada feedback, permitindo que o usuário selecione os que desejar; e criação semiautomática do texto de *análise de dogfooding,* com os feedbacks e os charters indicados na análise. Vejamos a seguir o detalhamento desse processo.

(1) Os charters são indexados na plataforma de RI *Apache Solr*, dados os problemas mencionados no capítulo anterior. O *Solr*[29] foi o software escolhido, pois fornece recursos robustos de indexação para realização de consultas textuais mais rápidas e precisas. Assim, foram criados 3 scripts, que serão nomeados de *charters_db*, *solr_module* e *charter_index*. Esses scripts são responsáveis, respectivamente, por: obter os dados de todos os charters de TEs; prover mecanismos para indexação e busca dos charters no *Solr*; e executar, mensalmente, os procedimentos de atualização da base de charters indexados no *Solr*. Com o uso desses scripts, os charters são recuperados e indexados no *Solr* para a realização das consultas;

(2) No protótipo, antes dos feedbacks serem classificados, eles passam por um método de *part-of-speech tagging* para identificação dos termos *substantivos* (i.e., NN*). Os termos *substantivos* são guardados na variável *nouns*, que é usada como parâmetro de consulta na recuperação dos charters[30] no *Solr*, utilizando a biblioteca *PySolr* (rever seção 4.2.2.3 e nota de rodapé 13). Esse processo é iterativo, ou seja, para cada feedback recuperado são identificados os substantivos (NN*) e estes são usados para buscar os charters. Os termos *substantivos* foram considerados pelo fato dos charters conterem informações dos componentes dos aparelhos (camera, phone, battery, etc.), que configuram *substantivos*. Para cada feedback relevante, o usuário poderá selecionar 1 ou mais charters recuperados. É através da seleção dos charters que o documento da *Análise de dogfooding* é criado. Essa análise consiste basicamente em agrupar os charters e seus feedbacks associados que devem ser considerados na realização das atividades de TEs. É criado, então, um documento textual contendo essas informações, que deverá ser reportado ao sistema de gerenciamento de atividades para disponibilização aos testadores de TEs.

(3) Durante a análise, é possível identificar feedbacks que estão relacionados aos mesmos charters. No entanto, a seleção de um charter para um feedback é independente da seleção desse mesmo charter para outro feedback. Em

[29] Solr: Plataforma de busca de código aberto escrita em Java, cujas principais funções são busca completa em textos e indexação em tempo real (Fonte: http://lucene.apache.org/solr/).

[30] Os charters contêm as informações dos componentes do aparelho (camera, tela, sensores, etc.) que os testes devem considerar. Para uma recuperação mais precisa dos charters diante dos muitos termos presentes nos feedbacks, consideramos a *remoção de stopwords* e uso de *dicionário de sinônimos* para os componentes que possuem múltiplos termos designativos.

outras palavras, cada feedback possui a sua própria lista de charters, podendo ser semelhante às sugestões de charters de outros feedbacks. No texto do documento de *Análise de dogfooding*, porém, é considerado apenas um charter, que aponta para os feedbacks que o referenciam.

As Figuras 26, 27 e 28 abaixo mostram, respectivamente, 2 feedbacks com os mesmos charters selecionados e o documento de *análise de dogfooding* resultante. Vejamos também os diagramas BPMN das Figuras 29, 30 e 31, que resumem o passo a passo dos procedimentos mencionados acima.

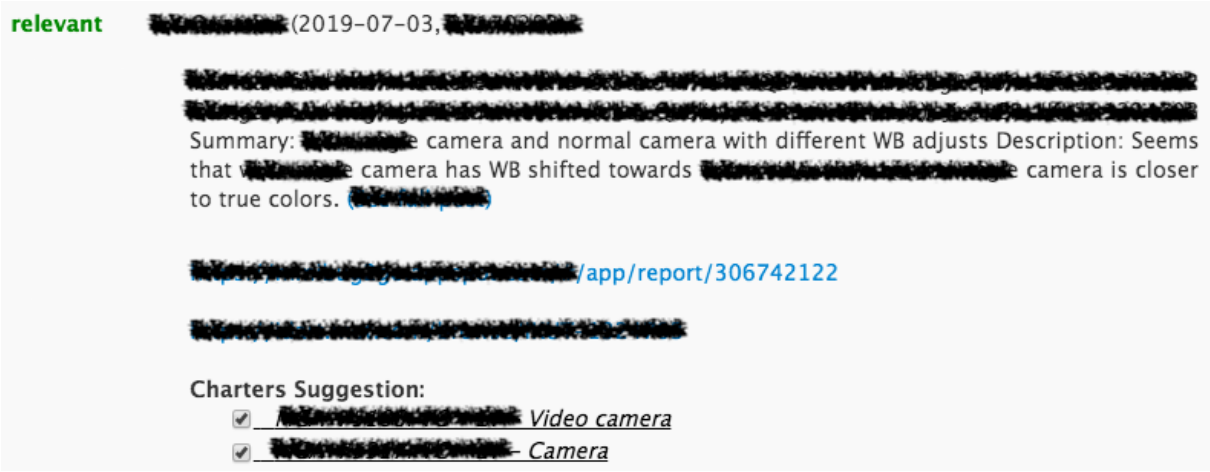

Figura 26: Feedback 1, com charters de *camera* e *video camera*
Fonte: Autor (2019)

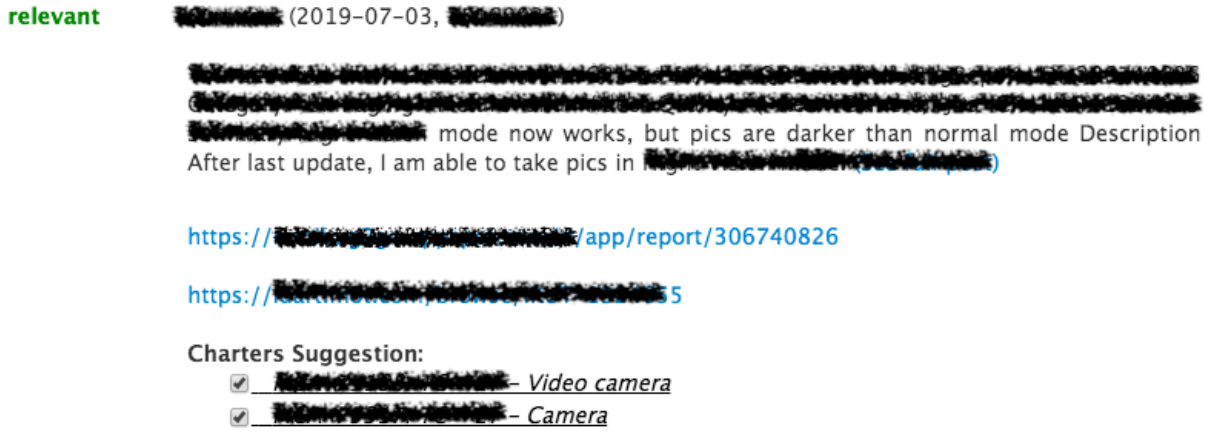

Figura 27: Feedback 2, com charters de *camera* e *video camera*
Fonte: Autor (2019)

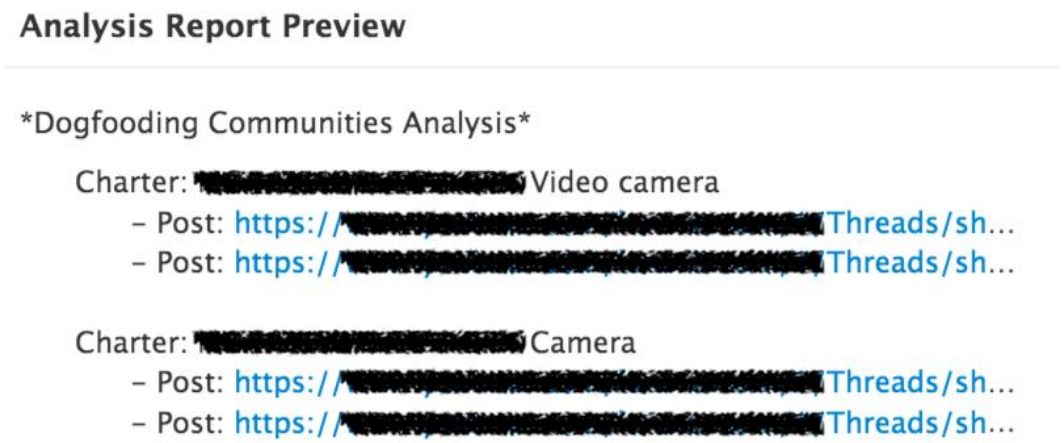

Figura 28: Documento de análise de dogfooding resultante
Fonte: Autor (2019)

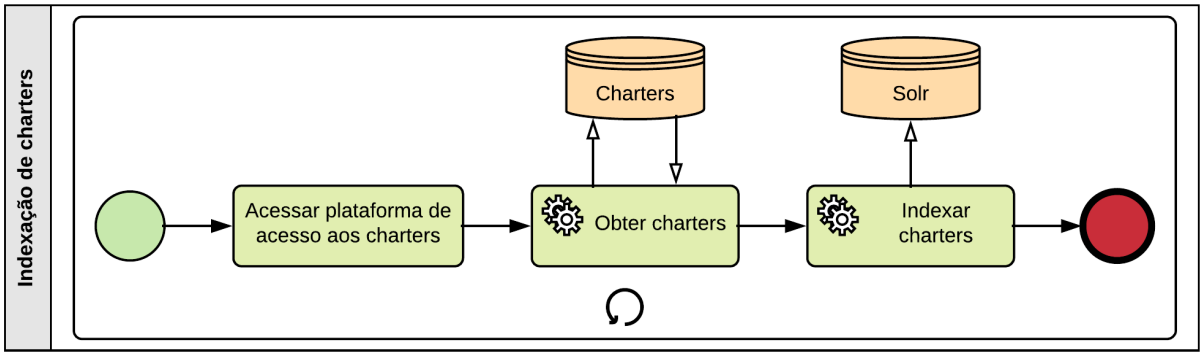

Figura 29: Diagrama BPMN do procedimento de Indexação de charters
Fonte: Autor (2019)

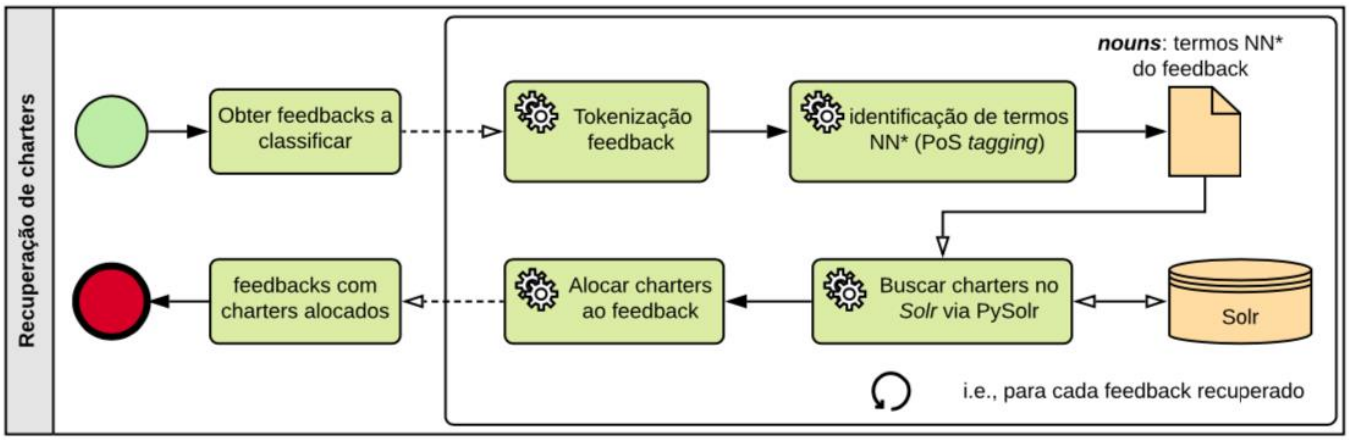

Figura 30: Diagrama BPMN do procedimento de Recuperação de charters
Fonte: Autor (2019)

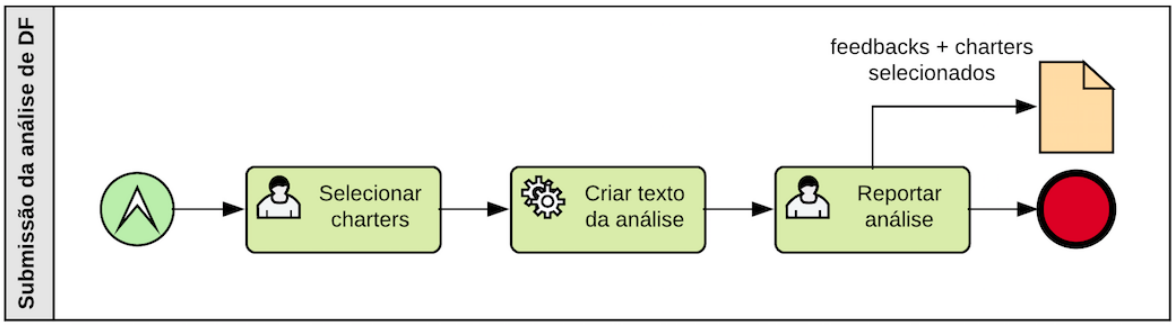

Figura 31: Diagrama BPMN do procedimento de Submissão da análise de DF
Fonte: Autor (2019)

Na Figura 30, é importante ressaltar que os processos de *tokenização* e *PoS tagging* não alteram o texto original dos feedbacks[31], servindo apenas para a busca e alocação dos charters para cada feedback.

Como dito antes (capítulo 4), nem sempre o protótipo recupera os charters apropriados para um dado feedback, podendo haver casos em que nenhum charter apropriado é recuperado. Isso pode acontecer por vários motivos: a base de dados de charters estar desatualizada ou os termos presentes nos feedbacks não são suficientes para encontrar os charters apropriados (i.e., quando ocorre o uso de diferentes termos para fazer referência à mesma funcionalidade do aparelho, ou ainda quando o feedback foi mal redigido). Esse último caso é tratável através de uma funcionalidade que foi posteriormente adicionada ao protótipo, a nível de interface gráfica (GUI): *Change charters*. Essa funcionalidade permite que o usuário faça buscas à base de charters no *Solr*, utilizando *tags* como parâmetro de busca.

Essas tags são criadas a partir das palavras dos sumários dos charters, com o uso de *expressões regulares* (regEx) e remoção de *stopwords*, de modo a resumir os sumários em poucas palavras. Por exemplo, para um charter com título *Test all sound system*, teríamos uma *tag* "*sound-system*" como resumo do título. A palavra *Test* é removida com uma regEx do tipo r'\b(?:test|et|tc)\b'[32], ignorando o *case* do texto (i.e., a regEx é aplicável para termos em caixa alta ou baixa), e a palavra *all* é removida através das *stopwords*, restando apenas as palavras *sound* e *system*, sendo estas concatenadas com o caractere "-". Para buscar/recuperar um ou mais charters, o

[31] Em termos de design OO (Orientação a Objetos), cada feedback é uma instância da classe Feedback. Além dos demais atributos (i.e., conteúdo, data, url, etc.) que são recuperados via SQL (Figura 21), o feedback também possui o atributo *charters*, que é preenchido com o resultado do processo de Recuperação de Charters da Figura 31.
[32] Essa regEx remove os prefixos dos títulos dos charters, que geralmente são *Test*, *TC ET* (i.e., *Exploratory Test Case*) ou apenas *ET* (*Exploratory Testing*).

usuário insere uma ou mais *tags*. Dessa forma, os charters relacionados às tags são recuperados e o usuário poderá atualizar a lista de charters do feedback escolhido.

5.5.1 Avaliação da Recuperação de Charters

A fim de verificar se os charters recuperados são relevantes para os feedbacks, foram realizados dois testes com 2 aparelhos diferentes. O objetivo foi verificar se, para cada feedback, pelo menos 1 dos charters da lista atende às especificações do feedback. Por exemplo, se o feedback menciona um problema de *camera*, espera-se que pelo menos 1 dos charters retornados atenda essa funcionalidade do aparelho. Esse teste foi realizado utilizando-se 40 feedbacks diferentes para cada aparelho. Vejamos o resultado a seguir, na Tabela 5.

	Aparelho 1	**Aparelho 2**	**Total**
n° de feedbacks com charters recuperados pelo protótipo	33	34	67
nº de feedbacks com charters recuperados via consulta	7	6	13

Tabela 8: Resultado da recuperação de charters
Fonte: Autor (2019)

Observando a tabela acima, vemos que o protótipo recuperou charters relevantes para aproximadamente 84% dos feedbacks de entrada (linha 1). A linha 2 mostra o número de feedbacks que não conseguiram recuperar charters, sendo necessário o uso da *consulta de charters*. Como já mencionado antes, esse último caso ocorre devido a várias causas: base de charters desatualizada (para funcionalidades novas dos smartphones); múltiplos termos designam a mesma funcionalidade (não havendo uma correspondência entre esses termos “sinônimos” no *Solr*; Feedbacks mal escritos. Esses casos, todavia, não são frequentes, como podemos ver nos resultados. Além disso, os usuários consultados aprovaram a ideia de realizar as consultas para recuperar charters nesses casos. Assim, podemos concluir que a funcionalidade da *consulta de charters* atingiu o objetivo proposto, com um nível aceitável de acurácia e usabilidade.

5.6 RETREINAMENTO (*Beta*[33])

Foi criada também uma funcionalidade (*beta*) com o objetivo de tentar melhorar o modelo de classificação, nos casos de falhas de predição. Inicialmente, surgiram duas questões:

(1) O modelo de classificação pode eventualmente atribuir labels erradas aos feedbacks. Seria interessante o usuário poder avaliar as classificações dos feedbacks retornados quando desejado, aprovando-as ou não. Essas avaliações serviriam como *input* para a realização de ajustes nos parâmetros dos modelos e/ou nos dados de treinamento;

(2) Os dados utilizados para treinar os modelos podem, com o passar do tempo, ficar obsoletos para classificar instâncias mais recentes. Se for considerada a natureza de softwares para dispositivos móveis, isso fica bem evidente, visto que as novas versões trazem novidades, mudanças e comportamentos não cobertos nos dados de treinamento anteriores. Dessa forma, os feedbacks avaliados em um período de uso da ferramenta poderiam ser utilizados para substituir os dados de treinamento menos recentes.

A ideia foi, então, adicionar um mecanismo que possibilitasse ao usuário avaliar as classificações, permitindo também o armazenamento de dados recentes (e pré-etiquetados manualmente), para a atualização dos modelos de classificação. Para isso, foi criado o script *retrain*. Esse script consiste em modificar o classificador corrente, substituindo exemplos antigos por exemplos novos e retreinando-o a partir deles. Essa funcionalidade também utiliza o *Solr* para armazenar os dados de retreinamento. Vejamos os detalhes dessa funcionalidade:

(1) O protótipo permite que os feedbacks[34] automaticamente classificados sejam avaliados pelo usuário. No nível de GUI, o usuário pode informar se a classificação automática do feedback está correta ou não. O feedback é então armazenado no *Solr*. Os feedbacks são salvos com seus conteúdos já pré-

[33] *Beta é* uma versão de um software ou produto em desenvolvimento, mas que é considerada aceitável para uso.

[34] O protótipo apresenta feedbacks *relevantes* ao usuário. No entanto, também é disponibilizada a opção *filter*, para que ele também possa acessar os feedbacks classificados como *irrelevantes*, podendo também avaliá-los.

processados[35], para facilitar sua utilização quando o classificador precisar ser retreinado. Os campos dos feedbacks armazenados são: *aparelho*, *conteúdo*, *classificação corrente*, *classificação do usuário* e *data*;

(2) O script *retrain* é executado periodicamente (a cada 30 dias), desde que haja uma quantidade mínima de 100 feedbacks novos armazenados no *Solr*. Como mencionado seção 4.3.1, a quantidade de dados para treinamento é 2691 feedbacks. Os 100 feedbacks novos de retreinamento irão substituir os 100 exemplos mais antigos desses 2691. Essa estratégia foi criada para que haja uma base de treinamento com feedbacks novos sem, no entanto, alterar o número de feedbacks de treinamento inicial. Vejamos o passo a passo:

(a) os dados de *treinamento* e *retreinamento* são recuperados;

(b) é realizada uma verificação nos dados de retreinamento para saber se os campos *classificação corrente* e *classificação do usuário* são divergentes. Se sim, serão consideradas apenas as classificações do usuário, visto que são as corretas a priori;

(c) os dados de *treinamento* são ordenados por *data* (do mais antigo para o mais recente), de modo a garantir que a *substituição dos dados* ocorra primeiramente para os mais antigos;

(d) na *substituição*, os dados de *retreinamento* são comparados com os de *treinamento*, através das labels (relevante/irrelevante). Um feedback de *retreinamento* substituirá um feedback de *treinamento*, desde que possuam a mesma label (ex.: substituição de um feedback relevante antigo por um relevante novo).

(e) é realizado um backup do arquivo de *treinamento* corrente (com versões), antes da substituição. Após isso, o arquivo é atualizado com os novos dados;

(f) ao final, o classificador é retreinado o com arquivo de *treinamento* atualizado.

[35] Os feedbacks salvos no Solr são normalizados apenas com os métodos textuais (remoção de stopwords, remoção de caracteres, etc.). No momento que o modelo está sendo retreinado, os métodos de transformação (tf-idf e seleção de atributos) são invocados.

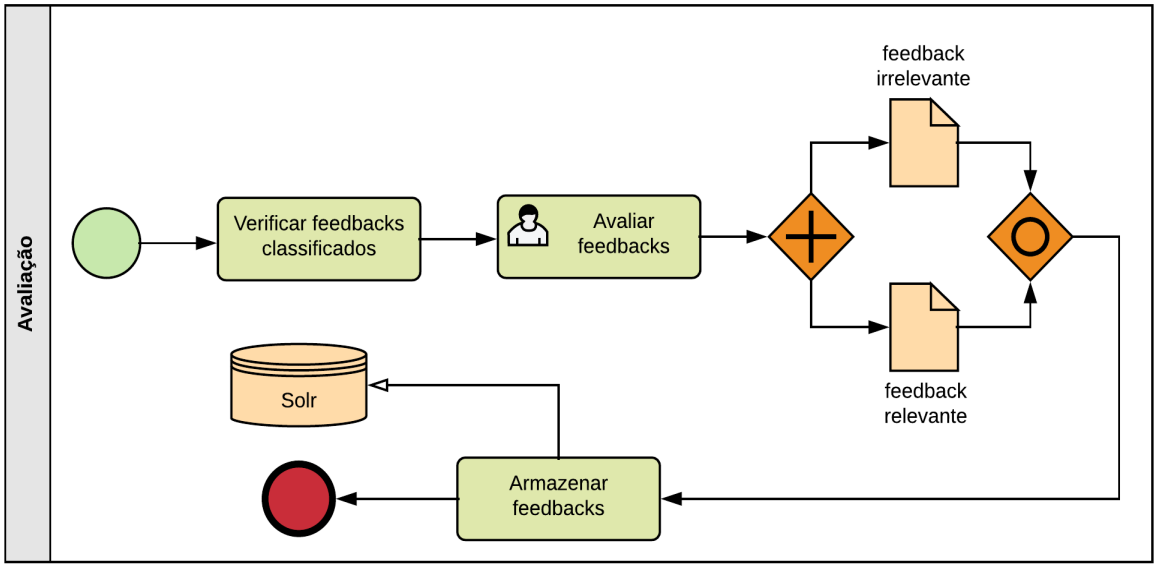

Figura 32: Diagrama BPMN do procedimento de Avaliação dos feedbacks
Fonte: Autor (2019)

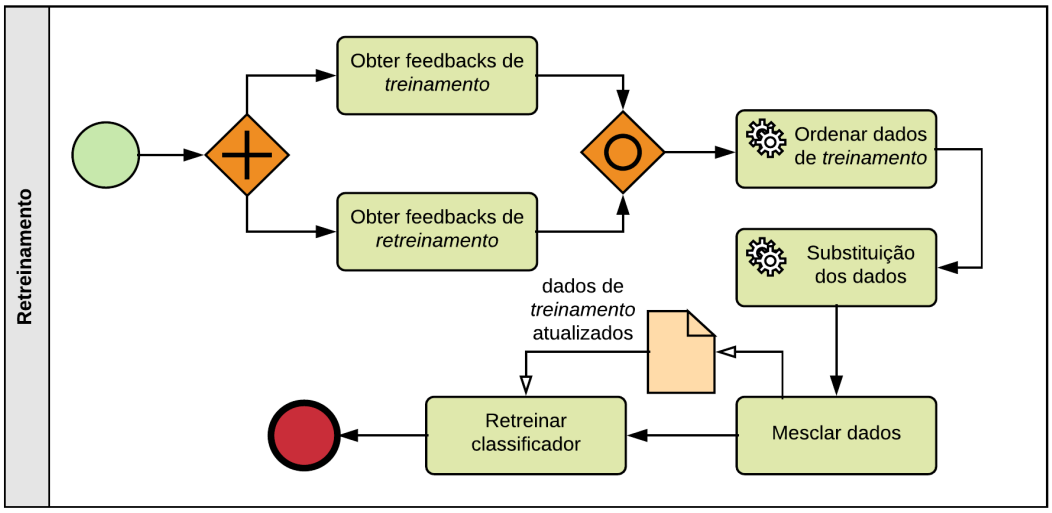

Figura 33: Diagrama BPMN do procedimento de Retreinamento
Fonte: Autor (2019)

Vale lembrar que a *etiquetagem* feita pelos usuários do protótipo não faz parte do fluxo principal do protótipo, porém sua utilização é reforçada caso os usuários comecem a perceber falhas de predição. Vejamos, na Figura 34, um exemplo de como o usuário pode avaliar o feedback.

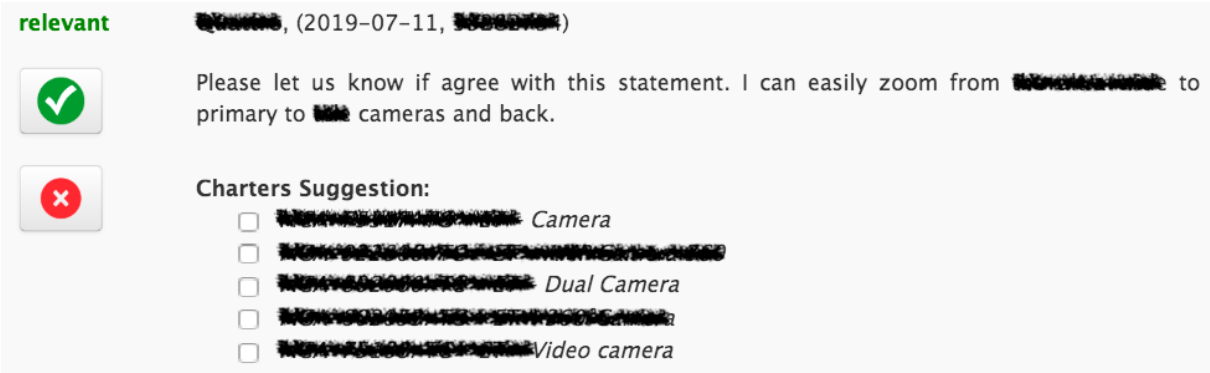

Figura 34: Avaliação da classificação dos feedbacks
Fonte: Autor (2019)

O usuário poderá clicar em um dos dois botões (verde e vermelho) para avaliar a classificação do feedback, e o feedback é diretamente armazenado no *Solr*. Essa funcionalidade foi implementada e está disponível para uso. No entanto, como foi pensada ao final do projeto de pesquisa, não restou tempo para realizar testes para validá-la apropriadamente. O número de feedbacks avaliados pelos usuários, até a entrega do protótipo, foi 54 amostras, e com apenas 11 amostras onde o classificador errou a classificação. Assim, a continuidade desta funcionalidade foi colocada como um *trabalho futuro*, no capítulo de Conclusão.

5.7 TESTES COMPARATIVOS DE USABILIDADE

Com o objetivo de averiguar os ganhos proporcionados com uso do protótipo nas *análises de dogfooding*, isto é, em relação aos procedimentos manuais, foram realizados *2 testes comparativos*.

5.7.1 Metodologia

Foram realizados 2 testes comparativos com 2 smartphones, onde cada teste compreendeu 2 *análises de dogfooding*: 1 *análise com o protótipo* e 1 *análise sem o protótipo*. Para isso, foram estabelecidos 4 critérios para a realização dos testes:

(1) 2 *análises de dogfooding*:

(a) uma análise manual e uma análise com o protótipo;

(b) 1 aparelho (smartphone):

Obs.: mesmo aparelho para as duas análises.

(c) 3 testadores;

(2) Mesmos *intervalos de data* (com e sem o uso do protótipo):

Ex.: utilizar o intervalo *01/06/2019* a *03/06/2019* nos dois casos;

(3) *Alternância dos testadores*:

(a) testadores diferentes realizam as análises com e sem o protótipo.

Obs.: o testador 1 realiza a análise usando o protótipo, num dado intervalo de data; o testador 2 realiza a análise manualmente, no mesmo intervalo de data.

(4) A a*nálise de dogfooding* compreende:

(a) recuperação/seleção dos feedbacks relevantes;

(b) seleção dos charters para os feedbacks;

(c) submissão da análise:

Obs.: o tempo gasto total de cada análise deve considerar todos os procedimentos citados.

Com os critérios definidos, foram alocados 3 testadores de TEs para realizar os testes comparativos. No protótipo, foram 10 feedbacks por análise, como o intervalo de data especificado. No caso das comunidades, como não há um número fixo de feedbacks a recuperar, os testadores analisaram 10 feedbacks aleatórios, dentro do mesmo intervalo de data utilizado no protótipo, e recuperaram os relevantes que conseguiram identificar. O número de feedbacks analisados nas comunidades foi reduzido devido os testes terem sido realizados num período em que o projeto tinha muitas demandas de trabalho, então foi necessário fazer algumas adaptações. Os testes foram realizados em 3 dias, para garantir um conjunto de feedbacks novos por teste: cada par de testadores (testador manual e testador com o protótipo) realizaram 4 análises por dia, isto é, 1 análise manual e 1 análise com o protótipo para cada um dos 2 smartphones. Enquanto um testador analisa os feedbacks manualmente, outro testador analisa-os usando do protótipo. Como dito antes, essa estratégia foi utilizada para que os feedbacks classificados pelo protótipo não “direcionasse” a decisão do testador quanto aos feedbacks que serão classificados manualmente. Vejamos os resultados nos quadros a seguir.

5.7.2 Resultados

	Smartphone 1			
Manual	testador 1	testador 2	testador 3	Tempo total (min)
Tempo gasto (min)	**15**	**14**	**11**	**40**
Feedbacks relevantes encontrados	**3 de 10**	**2 de 10**	**3 de 10**	Total de feedbacks relevantes: **8**
Protótipo	testador 2	testador 3	testador 1	Tempo total (min)
Tempo gasto (min)	**7**	**8**	**6**	**21**
Feedbacks relevantes retornados	**6 de 10**	**7 de 10**	**5 de 10**	Total de feedbacks relevantes: **18**
	Ganho de tempo	**47,50%**		
	Ganho de feedbacks relevantes identificados	**55,56%**		

Quadro 4: Teste de usabilidade com o Smartphone 1
Fonte: Autor (2019)

	Smartphone 2			
Manual	testador 1	testador 2	testador 3	Tempo total (min)
Tempo gasto (min)	**16**	**12**	**14**	**42**
Feedbacks relevantes encontrados	**7 de 10**	**8 de 10**	**7 de 10**	Total de feedbacks relevantes: **22**
Protótipo	testador 2	testador 3	testador 1	Tempo total (min)
Tempo gasto (min)	**6**	**5**	**4**	**15**
Feedbacks relevantes retornados	**7 de 10**	**8 de 10**	**7 de 10**	Total de feedbacks relevantes: **22**
	Ganho de tempo	**64,29%**		
	Ganho de feedbacks relevantes identificados	**0,00%**		

Quadro 5: Teste de usabilidade com o Smartphone 2
Fonte: Autor (2019)

Os resultados acima demonstram que o protótipo se saiu melhor que os procedimentos manuais de análise de dogfooding. No teste com o smartphone 1, do Quadro 4, os testadores conseguiram encontrar manualmente 8 feedbacks relevantes

dos 30 feedbacks totais. O tempo total gasto, considerando a relação dos charters e submissão da análise, foi de 40 minutos. Com o protótipo, foram obtidos 18 feedbacks relevantes dos 30 totais, gastando-se quase que metade do tempo dos procedimentos manuais, isto é, 21 minutos, também incluindo os charters e a submissão da análise. Em termos percentuais, os *ganhos* de tempo *e* de feedbacks relevantes identificados *foram, respectivamente,* 47.5% *e* 55.56%.

No teste com o smartphone 2, do Quadro 5, tanto com os procedimentos manuais quanto com o protótipo, foram obtidos 22 feedbacks relevantes, ou seja, não houve ganho no quesito *feedbacks relevantes*. Em todo caso, houve um *ganho de tempo* de 42 minutos para 15 minutos, que em termos percentuais foi de 64.29% para o protótipo.

Na prática, diferentemente do protótipo que retorna para o usuário os feedbacks relevantes explicitamente, a busca manual pelos feedbacks nas comunidades requer que o usuário analise vários deles para então poder selecionar os que de fato são relevantes, aumentando o tempo da atividade, mesmo com o intervalo de data especificado. É possível notar que, mesmo com o pequeno número de feedbacks analisados nas comunidades, o tempo necessário para obter os mesmos feedbacks que foram retornados pelo protótipo certamente iria aumentar. Isso mostra que os testadores dedicam pouco tempo para verificar os feedbacks, pois obter mais feedbacks relevantes requer mais tempo de análise, tornando-a pouco produtiva em termos manuais.

Também é possível verificar algumas diferenças nos resultados dos dois testes. Por exemplo, alguns dos testadores aparentaram insegurança quanto aos resultados do protótipo em relação às classificações e precisaram acessar as comunidades dos aparelhos para verificar os feedbacks, acessar o Jira para verificar os defeitos relacionados, etc. Os feedbacks do smartphone 1, um aparelho com mais tempo de teste, relataram defeitos um pouco mais sutis, diferente dos defeitos iniciais de um aparelho que acabou de ser submetido a testes (i.e., dos componentes básicos, como *camera*, *telefone*, *mensagens*, *ações*, etc.). Já o smartphone 2, um aparelho testado mais recentemente, não houveram dúvidas quanto à classificação dos feedbacks, por se tratarem de defeitos mais diretos. Isso responde o motivo pelo qual foi gasto um pouco mais de tempo na análise realizada no smartphone 1 em comparação com o smartphone 2.

Essa mesma ideia também se aplica à quantidade de feedbacks identificados, visto que no smartphone 1 foram identificados bem menos feedbacks com a análise manual do que com o uso do protótipo, diferente do smartphone 2, onde foram encontrados exatamente a mesma quantidade de feedbacks com e sem o uso do protótipo. Outro a ser considerado é experiência inicial com o uso do protótipo, pelo menos para alguns dos testadores, que pode ter impactado levemente nos resultados.

Em todo caso, é possível concluir que o protótipo proposto pode proporcionar um ganho significativo de produtividade nas atividades de *análise de dogfooding*. Dessa forma, o planejamento das atividades de TEs pode contar com mais feedbacks relevantes a considerar nos planejamentos, isto é, mais cenários potencialmente úteis para detecção de bugs não previstos intercalados com as diretrizes dos charters, e com uma diminuição de custo de tempo.

5.8 CONSIDERAÇÕES FINAIS

O objetivo deste capítulo consistiu basicamente no detalhamento do processo de desenvolvimento do protótipo, bem como na realização dos testes em relação às funcionalidades implementadas. Cada funcionalidade foi explicada individualmente, para garantir um melhor entendimento dos seus papéis no protótipo. Pudemos ver que os resultados de *treinamento* e de *classificação* foram satisfatórios, atendendo os objetivos deste trabalho.

Através dos procedimentos de *treinamento*, foi possível verificar que existem diferenças claras quanto as classes dos feedbacks, mesmo havendo casos em que os termos coocorram em feedbacks de classes diferentes. Verificamos também que um pequeno conjunto de atributos (i.e., termos mais representativos identificados pela Seleção de Atributos) foi suficiente para determinar a classificação correta da grande maioria dos feedbacks de teste, com poucos casos de falsos positivos e negativos.

Nos procedimentos de *classificação*, as matrizes de confusão mantiveram seus resultados bem distribuídos entre as classes (verdadeiros positivos e negativos), especialmente o SVM, que obteve a distribuição mais equilibrada. Sobre as métricas utilizadas na avaliação dos modelos, pudemos ver que todas elas obtiveram uma pontuação acima de 80%, demonstrando que os modelos conseguiram obter uma boa generalização em relação aos dados de treinamento.

Os testes estatísticos foram importantes para determinar o modelo ideal para o problema deste trabalho, visto que uma única execução de *testes de classificação* não oferece resultados consistentes em relação à qualidade dos modelos.

Nos testes de usabilidade, o protótipo se sobressaiu em relação procedimento manual, obtendo um ganho significativo tanto no aumento do número de feedbacks relevantes encontrados quanto na diminuição do tempo gasto nas análises. Esses testes, em particular, configuraram uma prova de conceito importante para o protótipo, pois o mesmo pode ser avaliado sob uma perspectiva prática de viabilidade de uso pelos próprios testadores.

6 CONCLUSÃO

Por fim, temos a conclusão do trabalho realizado. Aqui, constam as principais contribuições bem como a sugestão de trabalhos futuros. Como dito anteriormente, o principal objetivo deste trabalho foi desenvolver uma solução para automatizar parte dos procedimentos realizados durante as atividades de *análises de feedbacks de dogfooding*. Vejamos as seções a seguir.

6.1 PRINCIPAIS CONTRIBUIÇÕES

A primeira contribuição deste trabalho foi dar um enfoque mais profundo sobre a utilização prática da abordagem de Dogfooding, bem como mostrar sua importância dentro do campo de Testes de Software. Também deve ser levando em conta que esta pesquisa foi desenvolvida dentro de uma empresa real, que possui uma política interna de dogfooding.

De acordo com as revisões bibliográficas realizadas, percebeu-se que apesar de diversas empresas de software utilizarem efetivamente a abordagem *Dogfooding*, não foram encontradas iniciativas semelhantes à deste trabalho. Em outras palavras, não foram encontrados trabalhos que apresentam propostas relacionadas à utilização de mecanismos automáticos para obtenção de informações úteis, isto é, vindas de feedbacks de dogfooding. Dessa forma, a proposta trouxe como segunda contribuição um paradigma diferenciado em relação às atividades convencionais de análises de informações num tipo específico de artefato de software.

Como terceira contribuição, foi possível demonstrar que a utilização de mecanismos inteligentes de *classificação textual* tem um forte potencial para agilizar as atividades de testes baseados em informação, pois torna possível a obtenção de mais informações, antes ignoradas, com uma redução de tempo significativa, aumentando assim a qualidade das atividades realizadas. As informações ignoradas podem impactar negativamente a qualidade do produto final, visto que possíveis cenários de defeitos não são identificados. Além disso, através dos testes realizados na *classificação de feedbacks*, *recuperação de charters* e dos *testes de usabilidade*, foi possível comprovar os benefícios proporcionados pelo uso do protótipo em relação aos procedimentos manuais.

6.2 TRABALHOS FUTUROS

Como sugestões de trabalhos futuros, podemos pontuar as seguintes:

Outros encoders de texto: Neste trabalho, foi utilizado o encoder *TF-IDF*, que possui uma abordagem bem simples baseada em importância de termos. No entanto pode haver situações onde os dados dos feedbacks relevantes e dos irrelevantes estejam sobrescritos, dificultando a identificação de padrões claros que os separem. Dessa forma, como primeira sugestão, pode ser interessante a utilização de encoders mais eficientes, de modo a criar uma representação com características mais precisas dos textos de cada classe.

Retreinamento do classificador: Como foi dito, não restou tempo para realização de testes com essa funcionalidade. Assim, a sugestão é que sejam realizados os devidos testes para validação, os ajustes necessários, e estudos mais aprofundados para verificar a viabilidade de uso.

Ranking de charters: Foi explicado que, para cada feedback, há uma lista de charters. Porém, nem todos os charters estão relacionados às situações mencionadas nos feedbacks. Com base nisso, sugere-se que haja um critério de ordenação, de modo que os charters mais relacionados permaneçam no topo da lista. Em outras palavras, listar os charters do mais relacionado para o menos relacionado.

Sumarização de feedbacks: Alguns feedbacks, mesmos relevantes, podem conter excesso de informações. Desses feedbacks, apenas alguns trechos são úteis para direcionar os testes exploratórios. Seria interessante sumarizar os feedbacks, de modo a resumi-los a partir dos trechos realmente úteis.

Novos experimentos de usabilidade: os testes de usabilidade realizados avaliaram tanto a quantidade de feedbacks detectados e tempo economizado pelo protótipo em comparação com os procedimentos manuais. Em todo caso, também seria válido adicionar experimentos quanto aos charters sugeridos, de modo a verificar se a sugestão do protótipo proporciona ou não uma melhor cobertura de teste, comparando com os charters selecionados manualmente.

Implementação da proposta em outros ambientes: O trabalho considerou o ambiente do projeto CIn/Motorola. No entanto, a proposta pode ser aplicada em qualquer ambiente empresarial de produtos de software que utilizem feedbacks internos como fonte de informação para realização de testes exploratórios, como uma alternativa para agilizar os seus planejamentos.

REFERÊNCIAS

ABREU, J. C.; MARTINO, F. A.; SCHIAVONI, M. A. Testes Exploratórios em Software. **Revista Eletrônica Científica do CRA-PR,** v. 3, n. 1, p. 62-75, 2016.

ABUL-EZZ, S. **Dogfooding:** Conducting an Internal Beta Test for Your App. 2018. Disponível em <https://instabug.com/blog/dogfooding-conducting-internal-beta-test-app/>. Acesso em: 28 ago. 2018.

ANDRADE, P. H. M. A. **Aplicação de Técnicas de Mineração de Textos para Classificação de Documentos:** um Estudo da Automatização da Triagem de Denúncias na CGU. Dissertação (Mestrado Profissional em Computação Aplicada) - Universidade de Brasília. Brasília, 2015. 65 p.

ARANHA, C.; PASSOS, E. A Tecnologia de Mineração de Textos. **RESI - Revista Eletrônica de Sistemas de Informação**. v. 5. n. 2. Rio de Janeiro, 2006. Disponível em: <http://www.periodicosibepes.org.br/index.php/reinfo/article/view/171/66>. Acesso em: 08 jun. 2018.

BACH, J. **Exploratory Testing Explained**. v. 1.3. April, 2003. Disponível em: <http://www.satisfice.com/articles/et-article.pdf>. Acesso em: 14 mai. 2018.

_____.**Session-Based Test Management**. Software Testing and Quality Engineering magazine. November, 2000.

_____.**Where Does Exploratory Testing Fit?** Disponível em:<https://www.cmcrossroads.com/article/where-does-exploratory-testing-fit>. Acesso em: 16 mai. 2018

BAEZA-YATES, R. RIBEIRO-NETO, B. **Modern Information Retrieval**. New York: ACM Press, 1999.

BARION, E.; LAGO, D. Mineração de Textos. **Revista de Ciências Exatas e Tecnologia**. v. 3, n. 3. São Paulo, 2008.

BERRY, M. J. A.; LINOFF, G. **Data Mining Techniques:** For Marketing, Sales, and Customer Support. Wiley Computer Publishing. New York, 1997.

BORGES, H. B. **Redução de Dimensionalidade em bases de dados de Expressão Gênica.** Dissertação (Pós-graduação em Informática) – Pontifícia Universidade Católica do Paraná, PUC-PR. Curitiba, 2006.

BHATTI, K. GHAZI, N. **Effectiveness of Exploratory Testing**. Master Thesis (Software Engineering) – School of Engineering, Blekinge Institute of Technology. Ronneby, Sweden 2010. 79 p.

BRITO, E. M. **Mineração de Textos:** Detecção automática de sentimentos em comentários nas mídias sociais. Dissertação (Mestrado Profissional em Sistemas da

Informação e Gestão do Conhecimento) – Universidade Fundação Mineira de Educação e Cultura. Belo Horizonte, 2016.
CAETANO, C. **Testes Exploratórios de A a Z.** Disponível em: <http://www.linhadecodigo.com.br/artigo/1102/testes-exploratorios-de-a-a-z>. Acesso em: 25 mai. 2018

CAMILO, C. O.; SILVA, J. C. **Mineração de Dados: Conceitos, Tarefas, Métodos e Ferramentas**. Relatório Técnico. Goiás: Instituto de Informática da Universidade Federal de Goiás. Agosto, 2009. 28 p. n. RT-INF 001-09. Disponível em: <http://www.portal.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_001-09.pdf>. Acesso em: 17 set. 2018.

CARDOSO, O. N. P. Recuperação de Informação. In: *INFOCOMP* **Journal of Computer Science.** v. 2, n. 1. Lavras, 2004.

CARVALHO FILHO. **Mineração De Textos:** Análise De Sentimento Utilizando Tweets Referentes À Copa Do Mundo 2014. TCC (Bacharelado em Engenharia de Software) – Universidade Federal do Ceará. Quixadá, 2014. 46 p.

CARVALHO, R. C. **Aplicação de técnicas de mineração de texto na recuperação de informação clínica em prontuário eletrônico do paciente**. Dissertação (Mestrado em Ciência da Informação) – Programa de Pós-graduação em Ciência da Informação da Faculdade de Filosofia e Ciências - Universidade Estadual Paulista – UNESP, campus Marília. Marília, 2017. 201 p.

CARRILHO, J. **Desenvolvimento de uma Metodologia para Mineração de Textos**. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, PUC- Rio. Rio de Janeiro, 2007.

CASTANHEIRA, L. G. **Aplicação de técnicas de mineração de dados em problemas de classificação de padrões**. Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais. Belo Horizonte, 2008. 91 f.

COPELAND, L. **A practitioner's guide to software test design**. Norwood: Artech House, 2004.

CASSENOTE, M. R. S.; SECCO, A.; ANTONIAZZI, R. L.; CHICON, P. M. M. Aplicação de Mineração de Textos na Indicação de Palavras-chave em Artigos Científicos. In: Simpósio de Pesquisa e Desenvolvimento em Computação, 2., 2016, Cruz Alta. **Anais**… Cruz Alta: UNIVERSIDADE DE CRUZ ALTA - UNICRUZ, 2016. 10 p.

CENDÓN, B. V. **Ferramentas de busca na Web. Brasília:** Ciência da Informação. v. 30, n. 1. 2001. p. 39-49. Disponível em: <http://www.scielo.br/pdf/ci/v30n1/a06v30n1>. Acesso em: 21 set. 2018.

CHEN, H. **Knowledge management systems:** a text mining perspective. University of Arizona. Knowledge Computing Corporation, Tucson, Arizona, 2001. p. 18.

CRAIG, R.D.; JASKIEL, S. P., **Systematic Software Testing, Boston:** Artech House Publishers, 2002. Disponível em: https://www.twirpx.com/file/2461380/.

DAMASCENO, M. Introdução a mineração de dados utilizando o Weka. In: **V Congresso Norte-Nordeste de Pesquisa e Inovação** *(CONNEPI 2010).* Maceió, 2010. Disponível em <http://connepi.ifal.edu.br/ocs/index.php/connepi/CONNEPI2010/paper/viewFile/258/207>. Acesso em: 03 out. 2018.

DEFRANCO, M. **Not Eating Your Own Dog Food?** You Probably Should Be. 2014. Disponível em: <https://www.forbes.com/sites/michaeldefranco/2014/03/04/not-eating-your-own-dog-food-you-probably-should-be-2/#4826a45d692e>. Acesso em: 28 ago. 2018.

DOTTERWEICH, A. **What is Exploratory Testing?** Disponível em: <https://www.rainforestqa.com/blog/2017-08-04-what-is-exploratory-testing/>. Acesso em: 22 ago. 2018

EBECKEN, N. F. F.; LOPES, M. C. S; ARAGÃO C. M. C. Mineração de Textos. In: **Sistemas Inteligentes:** Fundamentos e Aplicações. Barueri: Manoele Ltda, 2003. p. 337-370.

EVANHAIM, D. **What Is 'Dogfooding' and How Can It Benefit Your Mobile App?** 2016. Disponível em: <https://blog.appsee.com/what-is-dogfooding-and-how-can-it-benefit-your-mobile-app/>. Acesso em: 28 ago. 2018.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**. v. 17, n. 3. American Association for Artificial Intelligence, 1996. Disponível em: <https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1230/1131>. Acesso em: 07 jun. 2018.

FELDMAN, R.; SANGER, J. **The text mining handbook:** advanced approaches in analyzing unstructured data. New York: Cambridge University Press, 2007. Disponível em: <https://bit.ly/2NKkqxZ>

FERREIRA, C. H. P. **Seleção de Atributos para Classificação de Textos usando Técnicas Baseadas em Agrupamento, PoS Tagging e Algoritmos Evolutivos**. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, UFABC. Santo André, 2016. 99 p.

GOMES, R. M. **Desambiguação de sentido de palavras dirigida por técnicas de agrupamento sob o enfoque da mineração de textos**. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-graduação em Engenharia Elétrica, Departamento de Engenharia Elétrica, PUC-Rio. Rio de Janeiro, 2009. 118 p.

GONÇALVES, T. et al. Analysing part-of-speech for portuguese text classification. In: **Computational Linguistics and Intelligent Text Processing.** [S.l.]: Springer, 2006. p. 551–562.

HARRISON, W. **Eating your own dog food**. v. 23, n. 3. IEEE Software, 2006. p. 5–7.

IEEE. **Guide to the Software Engineering Body of Knowledge**. v. 3.0. Tech. Rep. IEEE - 2014 version , 2014.

INDURKHYA, N.; DAMERAU, F. J. **Handbook of natural language processing**. Boca Raton: CRC Press, 2010. 2 ed. 2010. 702 p. Disponível em: <https://bit.ly/2NMMh0E>.

ITKONEN, J.; MANTYLA, M. V.; LASSENIUS, C. How do testers do it? An exploratory study on manual testing practices, In: **Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement** (ESEM 2009). Lake Buena Vista, Florida, USA. October, 2009. IEEE. p. 494-497.

ITKONEN, J.; RAUTIAINEN, K. Exploratory Testing: A Multiple Case Study. In: **Proceedings of the 4th International Symposium on Empirical Software Engineering** (ISESE 2005). Noosa Heads, Queensland, Australia. November, 2005. IEEE. p. 84-93.

JACKSON, P.; MOULINIER, I. **Natural language processing for online applications**: Text retrieval, extraction and categorization. [S.l.]: John Benjamins Publishing. v. 5. 2007.

KANER, C.; BACH, J.; PETTICHORD, B. **Lessons Learned in Software Testing**: A Context-Driven Approach. New York: Wiley Computer Publishing, 2002.

KANER, C.; FALK, J.; NGUYEN, H. Q. **Testing Computer Software**, New York: John Wiley & Sons, Inc., 1999.

KANER, C.; BACH, J.; PETTICHORD, B. **Lessons Learned in Software Testing,** New York: John Wiley & Sons, Inc., 2002.

KORDE, V.; MAHENDER, C. N. Text classification and classifiers: a survey. In: **International Journal of Artificial Intelligence & Applications** (IJAIA). v. 3, n. 2. 2012. 15 p.

KOWALCZYK, A. **Linear Kernel**: Why is it recommended for text classification? SVM Tutorial. 2014. Disponível em: <https://bit.ly/2II136E>. Acesso em 17 out. 2018.

LAFELDT, M. **The pros and cons of eating your own dog food.** Production Ready - Medium. 2017. Disponível em: <https://medium.com/production-ready/the-pros-and-cons-of-eating-your-own-dog-food>. Acesso em 08 ago. 2018.

LUCCA, G.; PEREIRA, I. A.; PRISCO, A.; BORGES, E. N. Uma implementação do algoritmo naïve bayes para classificação de texto. In: **IX Escola Regional de Banco de Dados - ERBD 2013.** Camboriú: Instituto Federal Catarinense - IFC. 2013. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/erbd/2013/0019.pdf>. Acesso em: 29 set. 2018

MADEIRA, R. O. C. **Aplicação de técnicas de mineração de texto na detecção de discrepâncias em documentos fiscais.** Dissertação (Mestrado em Modelagem

Matemática da Informação) – Fundação Getúlio Vargas, Escola de Matemática Aplicada. Rio de Janeiro, 2015. 66 p.

MÄNTYLÄ, M. V.; ITKONEN, J. **How Are Software Defects Found?** The Role of Implicit Defect Detection, Individual Responsibility, Documents, and Knowledge. *Information and Software Technology.* v. 56. p. 1597–1612.

MEDEIROS, E. A. **Técnica de Aprendizagem de Máquina para Categorização de Textos.** TCC (Graduação em Engenharia da Computação) - Escola Politécnica de Pernambuco, Universidade de Pernambuco. Recife, 2004. 67 p.

MORAIS, E. A. M.; AMBRÓSIO, A. P. L. **Mineração de Textos**. Goiás: Instituto de Informática da Universidade Federal de Goiás. 2007. 30 p. n. RT-INF 005-07.

MURPHY, K. P. **Machine Learning:** A Probabilistic Perspective. Massachusetts Institute of Technology. Cambridge, 2012. Disponível em: <https://www.dropbox.com/MachineLearning-ProbabilisticPerspective>.

NASEER, A. ZULFIQAR, M. **Investigating Exploratory Testing in Industrial Practice**. Master Thesis (Software Engineering) - School of Engineering, Blekinge Institute of Technology. Ronneby, Sweden 2010. 43 p.

OGURI, P. **Aprendizado de máquina para o problema de sentiment classification.** 2006. 54f. Dissertação (Mestrado em Ciências da Computação) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006. 54 p.

PACHECO, A. O problema de otimização. **Computação Inteligente: Máquinas aprendendo a solucionar problemas complexos**. Disponível em: <http://www.computacaointeligente.com.br/artigos/o-problema-de-otimizacao/>. 2015. Acesso em: 27 set. 2018.

_____. K vizinhos mais próximos – KNN. **Computação Inteligente: Máquinas aprendendo a solucionar problemas complexos**. Disponível em: <http://www.computacaointeligente.com.br/algoritmos/knn-k-vizinhos-mais-proximos/>. 2017. Acesso em: 27 set. 2018.

PASSINI, M. L. C. **Mineração de Textos para Organização de Documentos em Centrais de Atendimento.** Dissertação (Mestrado em Engenharia Civil) - Instituto Alberto Luís Coimbra de Pós-graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, UFRJ. Rio de Janeiro, 2012. 105 p.

PATEL, S. **Chapter 2:** SVM (Support Vector Machine) - Theory. Machine Learning 101 - Medium. 2017. Disponível em: <https://bit.ly/2pFAPFo>. Acesso em: 3 out. 2018.

PETROSKI, B. M. **Geração automática de casos de teste automatizados no contexto de uma suíte de testes em telefones celulares**. Monografia (Bacharelado em Ciência da Computação) - Universidade Federal de Santa Catarina. Florianópolis, 2009. 54 p. Disponível em: <http://www.labsoft.ufsc.br/publications/monografia_petroski.pdf>. Acesso em: 24 ago. 2018.

RASHMI, N.; SUMA, V. **Exploratory Testing:** An Overview. International Journal of Computer Applications, v. 131, n. 10, December, 2015.

RAHMAN, A. A. U.; HELMS, E.; WILLIAMS, L.; PARNIN, C. **Synthesizing Continuous Deployment Practices Used in Software Development**. 2015 Agile Conference, Washington, DC. 2015. p. 1-10.

REVISTABW. Aprendizado de Máquina: Aprendizado Supervisionado. **Revista Brasileira da Web:** Tecnologia. Disponível em: <http://www.revistabw.com.br/revistabw/aprendizagem-de-maquina-aprendizado-supervisionado/>. Acesso em: 01 ago. 2018.

REZENDE, S. O.; PUGLIESI, J. B.; MELANDA et al. Mineração de dados. In: **Sistemas Inteligentes: Fundamentos e Aplicações**. Barueri: Manoele Ltda., 2003. p. 307–335.

RODRIGUES, J. P. **Sistemas Inteligentes Híbridos para Classificação de Texto**. Dissertação (Mestrado em Ciência da Computação) - Centro de Informática da Universidade Federal de Pernambuco. Recife, 2009.

SANTOS, W. P. S. **Análise dos Tweets sobre a Black Friday através da Mineração de Texto e Análise de Sentimentos**. Projeto de Graduação (Bacharelado em Sistemas de Informação) - Universidade Federal do Estado do Rio de Janeiro, UNIRIO. Rio de Janeiro, 2016.

SANTANA, Rodrigo. **Cross Validation ou Obra do Acaso?** Minerando Dados. 2018. Disponível em: <https://bit.ly/2W59o7h>. Acesso em: 20 set. 2018.

_____. **Pipelines:** Como Automatizar Seus Processos de Machine Learning. Minerando Dados. 2018. Disponível em: <https://bit.ly/2Arkkni>. Acesso em: 23 set. 2018.

SEBASTIANI, F. **Machine learning in automated text categorization**. [S.l.]: ACM Computing Surveys. v. 34, n. 1. 2002, 47 p. Disponível em <http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf>. Acesso em 05 set. 2018.

SOARES, F. A. **Mineração de textos na coleta inteligente de dados na web.** Dissertação (Mestrado em Engenharia Elétrica) - Departamento de Engenharia Elétrica do Centro Técnico Científico da PUC-Rio. Rio de Janeiro, 2008.

SOARES, F. A. **Categorização automática de textos baseada em mineração de Textos**. Tese (Doutorado em Engenharia Elétrica) - Departamento de Engenharia Elétrica do Centro Técnico Científico da PUC-Rio. Rio de Janeiro, 2013.

SHOARAN, M. **Rule-Based Classifiers**. 2015. 23 slides. Disponível em: <https://web.uvic.ca/~maryam/DMSpring94/Slides/4_rules.pdf>. Acesso em: 24 abr. 2019.

TAN, A. H. Text mining: the state of the art and the challenges. In: **Proceddings..., PAKDD'99 workshop on Knowledge Discovery from Advanced Databases**, Beijing, 1999, p. 65-70.

THAKUR, A. The Oficial Blog of Kaggle.com. **Approaching (Almost) Any Machine Learning Problem**. Disponível em: <http://blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/>. Acesso em: 21 abr. 2019.

TICOM, A. A. M. **Aplicação das Técnicas de Mineração de Textos e Sistemas Especialistas na Liquidação de Processos Trabalhistas**. Dissertação (Mestrado em Engenharia Civil) - Instituto Alberto Luís Coimbra de Pós-graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, UFRJ. Rio de Janeiro, 2007. 101 p.

TINKHAM, A.; KANER, C. Learning Styles and Exploratory Testing. In: **Pacific Northwest Software Quality Conference (PNSQC) 2003**. Portland, Oregon, USA. October, 2003. p. 61-75

TINKHAM, A.; KANER, C. **Exploring Exploratory Testing**. 2003. Disponível em: <http://kaner.com/pdfs/ExploringExploratoryTesting.pdf>. Acesso em: 20 abr. 2018.

WEISS, S. M.; INDURKHYA, N.; ZHANG, T.; DAMERAU, F. J. **Text mining:** predictive methods for analyzing unstructured information. Sydney: Springer Science & Business Media, 2010. 237 p.

WILLET, P. **The Porter stemming algorithm:** then and now. Electronic Library and Information Systems, 40 (3). The University of Sheffield. Sheffield, 2006. p. 219-223.

WITTEN, I. H.; FRANK, E. **Data Mining:** Practical machine learning tools and techniques. 2 ed. San Francisco: Morgan Kaufmann, 2005. Disponível em: <https://bit.ly/1UIEtu2>. Acesso: 28 set. 2018.