UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

INSTITUTO DE COMPUTAÇÃO- ICOMP

PROGRAMA PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

UMA ABORDAGEM PARA DETECTAR RELATÓRIOS DE DEFEITOS DUPLICADOS BASEADA EM APRENDIZAGEM PROFUNDA

Thiago Marques Rocha

Manaus - AM

Agosto de 2020

Thiago Marques Rocha

UMA ABORDAGEM PARA DETECTAR RELATÓRIOS DE DEFEITOS DUPLICADOS BASEADA EM APRENDIZAGEM PROFUNDA

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, Campus Universitário Senador Arthur Virgílio Filho, como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador

Prof. Dr. André Luiz da Costa Carvalho

Universidade Federal do Amazonas - UFAM

Instituto de Computação- IComp

Manaus - AM

Agosto de 2020

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Rocha, Thiago Marques
R672a Uma abordagem para detectar relatórios de defeitos duplicados baseada em aprendizagem profunda / Thiago Marques Rocha . 2020
128 f.: il. color; 31 cm.

Orientador: André Luiz da Costa Carvalho
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. relatórios de defeito duplicado. 2. aprendizagem profunda. 3. mlp. 4. bert. 5. modelagem em tópicos. I. Carvalho, André Luiz da Costa. II. Universidade Federal do Amazonas III. Título

A Deus, a minha família e amigos.

Agradecimentos

A Deus em primeiro lugar por permitir que eu esteja realizando este sonho e vivenciando uma experiência gratificante.

A minha mãe Iris Almeida que não me deixou desistir nos momentos de tristeza e desânimo. Dedico a ela todo o meu agradecimento por estar sempre ao meu lado sendo o meu porto seguro.

Ao meu orientador professor André Carvalho que me incentivou e foi receptivo desde o início, gostaria de agradecer por todo apoio e dedicação, principalmente porque foi quem acreditou na ideia e viabilizou todo o caminho para que esta pesquisa fosse adiante.

Aos professores Fábio Santos e Maurício Figuereido que sempre me incentivaram, em especial ao professor Fábio que hoje é um grande amigo sempre compartilhando conhecimento e experiência.

Um obrigado especial a Ada Raquel quem revisou a monografia, contribuindo significativamente para a apresentação do trabalho.

Também gostaria de agradecer ao instituto SIDIA e à Universidade Federal do Amazonas por apoiar o progresso desta pesquisa, disponibilizando não somente o espaço e recursos, mas também incentivando o desenvolvimento científico.

Esta pesquisa, conforme previsto no Art. 48 do decreto nº 6.008/2006, foi parcialmente financiada pela Samsung Electrônica da Amazônia Ltda, nos termos da Lei Federal nº 8.387/1991, através de convênio nº 003/2019, firmado com o ICOMP/UFAM.

E a todos que direta e indiretamente fizeram parte da minha formação, o meu muito obrigado.

"Imagine uma nova história para sua vida e acredite nela."

(Paulo Coelho)

UMA ABORDAGEM PARA DETECTAR RELATÓRIOS DE DEFEITOS DUPLICADOS BASEADA EM APRENDIZAGEM PROFUNDA

Resumo

Em ambientes de desenvolvimento de software em larga escala, os relatórios de defeitos são mantidos por meio de sistemas de rastreamento de problemas e analisados por especialistas de domínio. Nesses sistemas, os usuários podem criar relatórios de defeitos de maneira despadronizada, ou seja, cada usuário pode relatar um problema específico com um conjunto exclusivo de palavras. Portanto, relatórios diferentes podem descrever o mesmo problema, gerando duplicação. Para evitar tarefas redundantes para a equipe de desenvolvimento, um especialista precisa examinar todos os novos relatórios enquanto rotula possíveis duplicatas. No entanto, essa abordagem não é trivial, nem escalável e impacta diretamente o tempo de correção dos defeitos. Esforços recentes para detectar relatórios de defeitos duplicados tendem a se concentrar em abordagens que utilizam redes neurais profundas que consideram as informações híbridas dos relatórios como recursos textuais e categóricos. Entretanto, essas abordagens ignoram que um único relatório pode ter várias duplicatas identificadas anteriormente e, portanto, várias descrições textuais, títulos e informações categóricas. Neste trabalho, propusemos o SiameseQAT, um método para detecção de relatórios de defeitos duplicados que considera não apenas informações sobre relatórios individuais, mas também informações coletivas de grupos de defeitos. O SiameseQAT combina aprendizado contextual e semântico com recursos textuais e categóricos, além de recursos baseados em extração de tópicos, utilizando a *Quintet Loss* uma nova função de perda introduzida por este trabalho, que considera o centroide de grupos duplicados e suas informações contextuais. Validamos nossa abordagem nos repositórios de *software* de código aberto Eclipse, NetBeans e Open Office, que incluem mais

de 500 mil relatórios de defeitos. Avaliamos a recuperação e a classificação de duplicatas, relatando uma média de Recall@25 de 71% para recuperação e 99% de AUROC para tarefas de classificação, resultados superiores aos apresentados por trabalhos relacionados.

Palavras-chave: relatórios de defeito duplicado, aprendizagem profunda, redes neurais profundas, aprendizagem semântica baseada em contexto, função de perda, quinteto, terceto, mecanismo de atenção, BERT, MLP, LDA, modelagem em tópicos.

UMA ABORDAGEM PARA DETECTAR RELATÓRIOS DE DEFEITOS DUPLICADOS BASEADA EM APRENDIZAGEM PROFUNDA

Abstract

In large-scale software development environments, defect reports are maintained through bug tracking systems and analyzed by domain experts. Since different users may create bug reports in a non-standard manner, each user can report a particular problem with a unique set of words. Therefore, different reports may describe the same problem, generating duplication. In order to avoid redundant tasks for the development team, an expert needs to look at all new reports while trying to label possible duplicates. However, this approach is neither trivial nor scalable and has a direct impact on bug fix correction time. Recent efforts to find duplicate bug reports tend to focus on deep neural approaches that consider hybrid information from bug reports as textual and categorical features. However, these approaches ignore that a single bug can have multiple previously identified duplicates and, therefore, multiple textual descriptions, titles, and categorical information. In this work, we propose SiameseQAT, a duplicate bug report detection method that considers not only information on individual bugs, but also collective information from bug clusters. The SiameseQAT combines context and semantic learning on textual and categorical features, as also topic-based features, with a novel loss function called *Quintet Loss*, which considers the centroid of duplicate clusters and their contextual information. We validated our approach on the well-known open-source software repositories Eclipse, NetBeans, and Open Office, that comprises more than 500 thousand bug reports. We evaluated both retrieval and classification of duplicates, reporting a Recall@25 mean of 71% for retrieval, and 99% AUROC for classification tasks, results that were significantly superior to related works.

Keywords: duplicate bug report, deep learning, deep neural networks, semantic context-based, loss function, quintet, triplet, attention mechanism, BERT, MLP, LDA, topic modeling.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

AP Aprendizagem Profunda

AM Aprendizagem de Máquina

BERT Bidirectional Encoder Representations from Transformers

bi-LSTM Bi-directional Long Short-Term Memory

BM25F$_{ext}$ BM25F estendido

BOW Bag of Words

CBOW Continuous Bag of Words

CNN Convolutional Neural Network

DMS Deep Siamese Model

DWEN Deep Word Embedding Neural Network

IDF Frequência Inversa de Documentos

tf-IDF Frequência do Termo na Frequência Inversa de Documentos

LDA Alocação Latente de Dirichlet

LSTM Long Short-Term Memory

MLP Multilayer Perceptron

OHE One-hot-encoding

QL Quintet Loss

RI Recuperação de Informação

RNN Recurrent Neural Network

SQAA SiameseQA-A

SQATA SiameseQAT-A

SQATW SiameseQAT-W

SQAW SiameseQA-W

SRP Sistemas de Rastreamento de Problemas

STA SiameseTA

STAT SiameseTAT

TL Triplet Loss

t-SNE Distributed Stochastic Neighbor Embedding

VSM Modelo de Espaço Vetorial

Glossário

Centroide - Representação média entre vetores.

CBOW - Método para prevê a palavra de contexto.

Embedding - Representação latente de características.

Mestre - Relatório de defeito representante de um grupo de defeitos.

Quintet Loss - Função de perda personalizada para tercetos de relatórios e centroides.

SiameseQA-A - Modelo com arquitetura siamesa, Quintet Loss (Q), Atenção (A) e Pesos iguais (A).

SiameseQA-W - Modelo com arquitetura siamesa, Quintet Loss (Q), Atenção (A) e Pesos treináveis (W).

SiameseQAT-A - Modelo com arquitetura siamesa, Quintet Loss (Q), Atenção (A), Tópicos (T) e Pesos iguais (A).

SiameseQAT-W - Modelo com arquitetura siamesa, Quintet Loss (Q), Atenção (A), Tópicos (T) e Pesos treináveis (W).

SiameseTA - Modelo com arquitetura siamesa, Triplet Loss (T) e Atenção (A).

SiameseTAT - Modelo com arquitetura siamesa, Triplet Loss (T), Atenção (A) e Tópicos (T).

Triplet Loss - Função de perda personalizada para tercetos de relatórios.

Skip-gram - Método para prevê as palavras que correspondem ao contexto da entrada.

Lista de símbolos

A	Relatório âncora aleatório
b	Constante definida como viés
C	Constante definida como margem
C_p	Centroide das duplicatas positivas de A
C_n	Centroide das duplicatas negativas de A
D	Dimensão vetorial
f	Função imagem
$f(x)$	Função imagem de x
k, n, m	Tamanho em unidade inteira
$L_{quintet}$	Função Quintet Loss.
N	Relatório negativo ao relatório A
P	Relatório positivo ao relatório A
Q_1	Função Triplet Loss definida como termo da equação da Quintet Loss.
Q_2	Função customizada para centroides como termo da equação da Quintet Loss.
V	Conjunto de palavras mais frequentes
v_i	Palavra top i mais frequente
T	Uma tripla composta por três relatórios $\{A, P, N\}$
t_i	Tempo t em um instante do tempo
W	Matriz de pesos

w	Vetor de pesos
X	Matriz numérica
x	Vetor numérico
Y	Combinação de x, w e b
Y_{true}, Y_{pred}	Matriz de classes
y_i	Vetor de classes na posição i
$SOFTMAX$	Função para representar a probabilidade dos dados serem de uma das classes definidas.
$DISTÂNCIA$	Função de similaridade entre dois vetores.
$COSSENO$	Função de similaridade do cosseno entre dois vetores.
MAX	Função para retornar o maior valor entre números.

Sumário

1 Introdução

Os sistemas de rastreamento de problemas (SRP) são ferramentas essenciais usadas para coordenar as correções de defeitos em grandes ambientes de desenvolvimento de *software*. Essas ferramentas permitem que os relatórios de problemas sejam provenientes de diferentes fontes, como desenvolvedores, testadores e usuários em geral, podendo apresentar um nível inconsistente de detalhes (AGGARWAL et al., 2017). Normalmente, o processo de triagem de defeitos pode ser dividido em três fases, conforme explicado por Zhang et al. (2016): (i) entendimento, (ii) seleção e (iii) correção. Compreender (i) e selecionar (ii) um relatório de defeito não são tarefas triviais, pois diferentes usuários escrevem relatórios com suas próprias palavras, gerando um custo adicional para o *triager*, um especialista capaz de entender e separar esses documentos. O *triager* deve entender todos os relatórios publicados em um dia enquanto procura relatórios duplicados para rotular, evitando retrabalho (UDDIN et al., 2017). Nesse sentido, a correção (iii) é a última fase da triagem, em que a equipe de desenvolvimento corrige o possível defeito relatado.

A partir do nosso levantamento bibliográfico, identificamos que desde 2007 a comunidade científica investiga formas diferentes de detectar relatórios duplicados, seja através de métodos manuais ou, mais recentemente, de métodos automáticos, onde relatórios são normalmente representados vetorialmente de forma que os duplicados estejam próximos, facilitando a recuperação de possíveis duplicatas. Entretanto, detectar relatórios de defeitos duplicados enfrenta desafios como jargões de projeto, bases desbalanceadas, conteúdo de linguagem não natural, entre outros.

Nesse contexto, como uma maneira de automatizar a detecção de duplicatas e reduzir o tempo gasto nos processos de triagem, propomos o SiameseQAT, uma abordagem baseada na topologia siamesa de redes neurais para detectar relatórios de defeitos duplicados, combinando em uma mesma arquitetura mecanismos de atenção, informação de tópicos latentes e a função de perda *Quintet Loss*.

O restante deste capítulo está organizado da seguinte maneira: na Seção

1.1, levantamos as motivações desta pesquisa. Na Seção 1.2, descrevemos o problema destacando as diferenças entre triagens de defeitos, além da deteccão de duplicatas. Na Seção 1.3, contextualizamos as diferentes abordagens para detectar duplicatas, enquanto, na Seção 1.4, apresentamos a nossa abordagem proposta. Na Seção 1.5, apresentamos a hipótese deste trabalho. Na Seção 1.6, detalhamos os objetivos propostos. Na Seção 1.7, apresentamos as contribuições da pesquisa. Por fim, na Seção 1.8, esclarecemos a organização desta dissertação.

1.1 Motivação

Os relatórios de defeitos são documentos criados por SRPs, ferramentas que mantêm muitos relatórios que descrevem defeitos de *software*. Para cada relatório de defeito, existem informações textuais e não textuais preenchidas por usuários diferentes, como desenvolvedores, testadores ou qualquer outro usuário (AGGARWAL et al., 2017). Informações textuais são todos os atributos, como o título e a descrição do problema, que descrevem o defeito. Informações não textuais são todos os atributos numéricos ou categóricos que descrevem o *software* ou o defeito, como produto, componente, prioridade, severidade, resolução, *status*, e versão.

Cada relatório de defeito duplicado recebe um rótulo manual fornecido por um *triager*, que insere o ID de um relatório relacionado ao mesmo problema (ZHANG et al., 2016). Quando dois relatórios de defeito são marcados como duplicados, ambos estão descrevendo o mesmo defeito usando ou não o mesmo vocabulário. O relacionamento entre relatórios duplicados pode ser visualizado em uma estrutura de árvore, de modo que o primeiro relatório (o nó raiz) é conhecido como relatório mestre e os demais (os nós filhos) como duplicatas, conforme mostrado na Figura 1.

Para detectar relatórios de defeitos duplicados, um algoritmo precisa produzir padrões de representação sobre defeitos duplicados, de maneira que identifique as características do conteúdo duplicado e a semelhança entre os documentos. No entanto, há cinco características dos relatórios de defeitos, apresentados na Tabela 1, que são responsáveis pelos principais desafios na

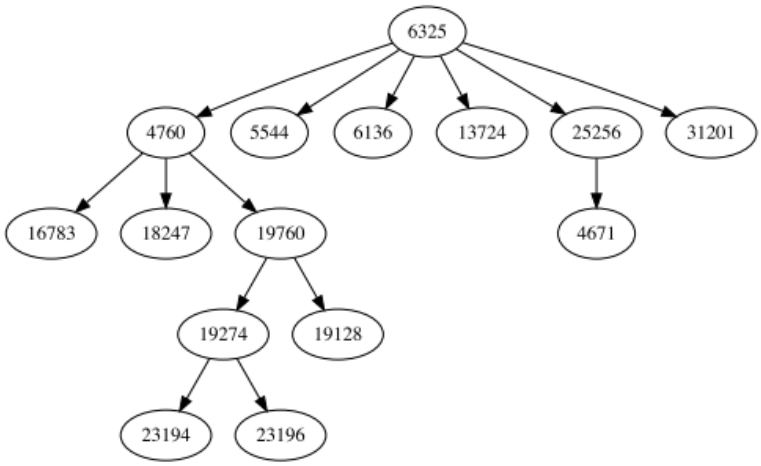


Figura 1 – Um exemplo de relatórios duplicados. O relatório #6325 é o mestre e todos os demais são relatórios duplicados.

Fonte: Sadat et al. (2017)

detecção de duplicatas e são descritas a seguir:

1. *Vocabulário de projeto*: os usuários de projetos diferentes geralmente têm convenções diferentes quando relatam defeitos de *software*, usando palavras-chave de domínio, como *word*, *doc*, *Eclipse* ou *Java*.

2. *Linguagem natural*: os relatórios de defeitos geralmente são escritos em linguagem natural descrevendo o ocorrido. No entanto, não há total confiança de que o texto tenha sido digitado sem conteúdo não natural como stack traces, códigos-fonte entre outras evidências de desenvolvimento.

3. *Linguagem não natural*: desenvolvedores e testadores geralmente anexam *stack traces*, *logs*, etapas para reproduzir o problema e outras evidências para ajudar a entender o defeito do *software*. Esses dados são adicionados no campo, descrição, com o que é escrito em linguagem natural.

4. *Variação semântica/lexical*: diferentes formas de se referir a um mesmo assunto, como, por exemplo: "OO", ou "OOo", que faz menção ao *Open Office*; *NullPointerException* e *NP*, que significa referência nula no tempo de execução; *FS* que significa *File System*.

5. *Bases desbalanceadas*: a proporção de relatórios de defeitos duplicados em um projeto pode ser baixa em comparação com relatórios não duplicados, como demonstrado na Tabela 9 no Capítulo 4. Isso caracteriza um problema clássico de dados desbalanceados em tarefas de classificação e recuperação.

Para enfrentar esses desafios, propusemos a abordagem SiameseQAT para detectar relatórios de defeitos duplicados em tarefas de classificação e recuperação. Nossa abordagem pode aprender diversas características de relatórios de defeitos a partir de entradas textuais e categóricas, eliminando a necessidade de engenharia manual para descobrir padrões de representação, resolvendo os desafios 1, 2 e 3. Além disso, o SiameseQAT extrai o significado semântico e de contexto dos recursos de defeitos através de vetores *embeddings* e extração de tópicos das bases de relatórios de treino, solucionando o desafio 4. Para superar o problema dos dados desbalanceados, nossa abordagem usa a estratégia siamesa para treinar com entradas equilibradas, através de um relatório aleatório, um positivo, um negativo e seus centroides ao mesmo tempo. Mais detalhes relacionados aos pré-requisitos da nossa abordagem referente à aprendizagem profunda, rede siamesa, funções de perda e vetores *embeddings* serão descritos no próximo capítulo desse trabalho.

1.2 Definição do Problema

Se não existisse uma etapa preliminar para analisar e identificar os relatórios duplicados, um desenvolvedor começaria a trabalhar na correção de um defeito sempre que o mesmo fosse reportado no SRP. Por essa razão, o time de desenvolvimento elege um especialista que seleciona os defeitos duplicados e os rotula um a um, evitando retrabalho no projeto. O processo de triagem pelo especialista, apesar de eficiente, não é escalável, visto que centenas de relatórios podem ser enviados diariamente para um determinado projeto dependendo do seu escopo. Além disso, ainda existe uma não padronização da informação no conteúdo dos relatórios (LI et al., 2018).

Identificar relatórios duplicados não é a única tarefa na triagem de

Tabela 1 – Exemplos de relatórios de defeitos duplicados para três projetos de *software* de código aberto.

Project	Product	Duplicate descriptions
Eclipse	WTP	**Title**: Deleting a validation-disabled resource can lead to NPE. **Description**: If a resource is disabled from validation and then removed, a NPE... It most likely occurs because IContainer#findMember() can return a null... in org.eclipse.wst.validation.internal.DisabledResourceManager#load(IProject) line 91...
	Common Tools	**Title** : NPE in DisabledResourceManager.save(). **Description** : In the context of 424340 we faced a NPE in DisabledResourceManager.save() java.lang.NullPointerException at org.eclipse.wst.validation.internal.DisabledResourceManager.save(DisabledResource...
NetBeans	Platform	**Title** : Filesystem's state could be changed to read-only when I have file lock. **Description** : How to reproduce: Let have a read-write FS. Open some file from it into the editor. Edit file but do not save it. Set FS state to read-only. ERROR - FS state could be changed and file could be saved although FS is read- only.
		Title : Saving (CTRL-S or File/Save) a file on read-only filesystem causes...IOException. Save All action correctly reports that it can't save file. **Description** : java.io.IOException: File Outer/Inner/Pokus.java is readonly. at org.openide.text.EditorSupport$5.run(EditorSupport.java:422) at org.openide.filesystems.FileSystem.runAtomicAction(FileSystem.java:357) at org.openide.text.EditorSupport.saveDocument(EditorSupport.java:412) at com.netbeans.developer.modules.loaders.ja...
Open Office	Writer	**Title** : File saved as M.Word 97/2000/XP (.doc) corrupted. **Description**: The file came to me as a one page .doc form (NOT .dot) ... as Word 97/200/XP. The resulting file does not open in Word 2000 and ... in OO Writer ... tables are missing and replaced by a lot of space forcing the final text on to the next page...
		Title : WW8: table with center alignment disappears when exported. **Description** : Hi My configuration : OOo 3.0.1 FR under Ubuntu 8.04... Step to reproduce : 1/ open new text document ; 2/ insert a table, for example ... 4/ choose Alignment Center and click Ok ; 5/ save the file as odt 6/ menu File > Save ... MS-Word 97/2000/XP (.doc) 7/ close the file .doc 8/ reopen the doc file with OOo 3.0.1...

defeitos. Nesse sentido, a triagem de um defeito é dividida em três grandes fases: entendimento, seleção e correção do defeito. A Figura 2 descreve em alto nível as tarefas para cada fase. Na primeira fase, **entendimento**, um especialista tenta compreender o defeito por meio das informações no documento de defeito. Na segunda fase, **seleção**, o especialista avalia e separa os defeitos reportados por contexto do produto, módulo ou componentes, produzindo ou não novas atividades para o time de desenvolvimento corrigir, caso o defeito seja autêntico. Por fim, durante a fase de **correção**, um desenvolvedor poderá avaliar e corrigir o defeito pendente de solução.

Detectar relatórios duplicados é uma atividade praticada durante a segunda fase da triagem. Nesse sentido, nossa pesquisa investiga o processo de triagem sobre defeitos duplicados, identificando quando dois ou mais relatórios descrevem, ou relatam o mesmo problema. Essa atividade faz parte da fase de *seleção*, como explicam Zhang et al. (2016).

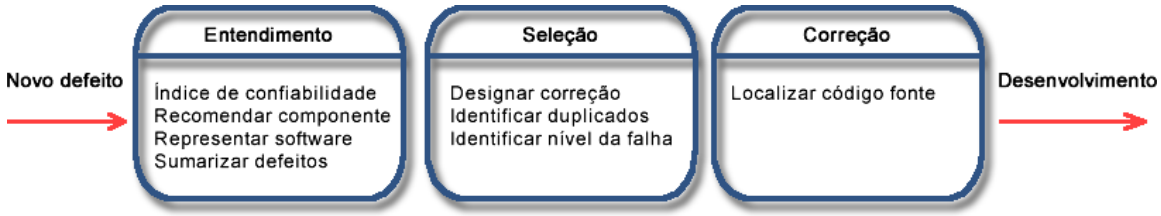


Figura 2 – Fases da triagem de um defeito.

Fonte: Adaptada de Zhang et al. (2016)

1.3 Contexto

Como mencionado, o trabalho de triagem não é facilmente escalável em uma situação de aumento de relatórios pendentes de análise. Assim, soluções que automatizam os processos de triagem, podem impactar diretamente os custos de gerenciamento de *software*, oferecendo a possibilidade de auxiliar na tomada de decisões e evitar o retrabalho do desenvolvimento durante a correção do defeito. Muitos pesquisadores exploraram abordagens automáticas para detectar relatórios duplicados, que podem ser divididas em métodos baseados em recuperação de informações (RI) e aprendizagem de máquina/aprendizagem profunda (AP). Geralmente, os recursos de texto são extraídos de campos como

título e descrição para gerar um documento de texto sobre os relatórios. De outro modo, as soluções híbridas propõem o uso de recursos de texto, além de categorias como produto, componente, prioridade, severidade, resolução, *status*, e versão e, em alguns casos, informações de execução, como *stack trace* ou *logs*.

Os métodos baseados em RI geralmente implementam variações dos modelos de espaço vetorial (VSM) (SUN et al., 2011; NGUYEN et al., 2012; KLEIN; CORLEY; KRAFT, 2014; ROCHA et al., 2016; XUAN et al., 2017) para representar relatórios de defeitos com base em padrões como frequência de palavras ou distribuição de tópicos, extraindo sintaxe e contexto dos relatórios. Por outro lado, os métodos mais recentes, baseados em aprendizagem de máquina, tendem a se concentrar em redes neurais profundas (BUDHIRAJA et al., 2018a; DESHMUKH et al., 2017), usando técnicas como *Multilayer Perceptron* (MLP), *Convolutional Neural Network* (CNN), *Long Short-Term Memory* (LSTM) ou *Bidirectional Long-Short Term Memory* (BiLSTM). Esses modelos extraem padrões latentes do conteúdo textual e categórico dos relatórios, manifestando uma maneira diferente de extrair padrões latentes de relatórios de defeitos, representando os documentos segundo seu contexto semântico e sintático.

Recentemente, os métodos baseados em AP alcançaram resultados superiores em comparação com os métodos baseados em RI (BUDHIRAJA et al., 2018a). Ainda mais cedo, Deshmukh et al. (2017) obtiveram resultados muito melhores, demonstrando que os recursos textuais e categóricos têm mais impacto juntos. Os resultados revelaram um enorme valor de precisão para duas tarefas, classificação e recuperação das duplicatas.

1.4 Abordagem

Embora os trabalhos relacionados tenham alcançado resultados satisfatórios na tarefa de detecção de relatórios de defeitos duplicados, acreditamos em melhorias na detecção explorando três novos componentes em uma única abordagem para a representação dos relatórios: a representação textual baseada em atenção (DEVLIN et al., 2018); a representação do contexto do relatório de defeito usando modelagem de tópicos (NGUYEN et al., 2012); e a utilização

de informações sobre grupos de duplicadas para melhorar a representação aglomerada das duplicatas.

Representação textual baseada em atenção: propomos o uso de mecanismos de atenção em vocabulários de relatórios, enquanto métodos anteriores utilizaram abordagens como CNN, LSTM e BiLSTM, redes neurais conhecidas como recorrentes e espaciais para processamento textual sem o viés de atenção. Nessas arquiteturas, o contexto das palavras não é ponderado de forma individual, ocasionando em representações que descrevem a sentença dos documentos e não o contexto das palavras. Nesse sentido, acreditamos resolver desafios enfrentados nesta tarefa, como a variação semântica entre as palavras, e o processamento de linguagem natural e não natural, onde modelos sem mecanismo de atenção ignoram esses cenários. Portanto, uma rede baseada em atenção entenderá o contexto maior das sequências ao analisar a semântica de cada termo, facilitando a detecção de mudança de contexto no conteúdo textual. Na literatura, as Representações de Codificação Bidirecional através de Transformadores ou em inglês, *Bidirectional Encoder Representations from Transformer*s (BERT), transformaram-se em modelos estado-da-arte para representação textual, justamente por aprenderem contextos das palavras. Em trabalhos como o de Zafar et al. (2019), a detecção de contextos de mensagens em repositórios de *software*, construiu representações textuais melhores sobre as mensagens de desenvolvedores demonstrando o impacto do mecanismo de atenção através da BERT. Dessa maneira, acreditamos melhorar a representação sobre relatórios de defeitos incorporando a mesma estratégia no aprendizado textual.

Contexto textual usando modelagem de tópicos: também propomos o aprendizado de padrões sobre relatórios de defeitos com base em recursos de modelagem de tópico, como uma forma adicional de caraterizar os relatórios, melhorando a representação textual em relatórios de defeitos por meio da distribuição de tópicos extraída para cada relatório cobrindo um entendimento global da base de dados de relatórios. O uso destes métodos captura os tópicos compartilhados entre os relatórios e pode aumentar a precisão da detecção (NGUYEN et al., 2012). Portanto, esperamos gerar representações semânticas com a modelagem baseada em tópicos em uma arquitetura de aprendizagem

profunda, para demonstrar o impacto dessa representação no aprendizado de padrões sobre relatórios.

Usando informações sobre grupos de duplicatas para melhorar a representação dos relatórios: pretendemos, através de uma modificação algébrica na função de perda baseada em tercetos, a *Triplet Loss* (TL), impulsionar o aprendizado de características sobre relatórios de defeitos. Segundo Schroff et al. (2015a), a TL é tipicamente utilizada em problemas para detecção de similaridade entre objetos, onde se têm um exemplo âncora em conjunto com um exemplo positivo e negativo, e assim treina-se uma rede para discriminar a similaridade entre as instâncias. Dessa maneira, primeiro investigaremos o uso de informações sobre os grupos de duplicatas, para treinar esse conjunto de tercetos em conjunto com o centroide do grupo destes exemplos, prescrevendo assim, uma nova função de perda para redes neurais, a Quintet Loss, e dessa maneira avaliar o impacto dessa estratégia na representação dos relatórios. Conforme apresentado por Ge (2018) e Wu et al. (2017), a TL pode induzir tercetos com igual importância no treinamento por causa do mesmo peso usado para todos os tercetos, e também pode incluir os tercetos que não representam a distribuição de dados global, pois geralmente são selecionados por amostragem. Nesse sentido, uma hipótese é que o uso de informações coletivas sobre o grupo de duplicatas pode ajudar a adaptar melhor as duplicatas no espaço latente em redes neurais, reduzindo a importância igual de tercetos no treinamento, minimizando as sobreposições de conjuntos duplicados e, consequentemente, melhorando as tarefas de discriminação sobre duplicatas.

Portanto, os três componentes da abordagem proposta são idealizados em uma única arquitetura siamesa, semelhante ao que foi feito por Deshmukh et al. (2017), para também os receber os recursos textuais e categóricos de relatórios de defeitos, além da nova entrada de recursos de tópicos, aprendendo representações contextuais semânticas sobre os documentos para duas tarefas de detecção, recuperação e classificação das duplicatas.

1.5 Hipótese

A hipótese definida para este trabalho considera que: "melhorar a representação de relatórios de defeitos através de técnicas mais avançadas como aprendizagem profunda e adição de novas informações relativas ao contexto do relatório pode melhorar a representação dos documentos para impactar em tarefas relacionadas à detecção de relatórios duplicados."

1.6 Obejtivo Geral

Modelar e avaliar abordagens baseadas em aprendizagem profunda para detectar relatórios de defeitos duplicados utilizando informações textuais, de tópicos, categóricas e que considerem informações oriundas de agrupamentos de relatórios duplicados para melhorar a qualidade em tarefas de recuperação e classificação de relatórios duplicados.

1.6.1 Objetivos Específicos

Os objetivos específicos são:

1. Quantificar o impacto que informações de outros relatórios, como tópicos e informações sobre grupos de duplicatas, para impactar na detecção de relatórios de defeitos duplicados;

2. Avaliar o efeito do uso de técnicas baseadas em modelos de atenção para melhorar a representação do texto presente em um relatório de defeito;

3. Avaliar o impacto individual de cada alteração e modelar um método que unifique as melhores práticas identificadas para a detecção de relatórios de defeitos duplicados.

1.7 Contribuições

As principais contribuições do nosso trabalho são:

1. Uma nova função de perda, a *Quintet Loss*, que utiliza centroides de grupos duplicados enquanto gera seu espaço latente sobre relatórios de defeitos, aproximando instâncias semelhantes e construindo grupos mais coesos de duplicatas.

2. A implantação de aprendizagem semântica baseada em contexto por meio da modelagem de tópicos e de modelos de atenção em conjunto com uma arquitetura siamesa para gerar relações contextuais em recursos de texto e categóricos a partir de relatórios de defeitos.

Os resultados desta dissertação foram compilados em um artigo científico submetido à revista IEEE Access, cuja revisão ainda não foi finalizada.

1.8 Organização do Trabalho

Esta dissertação está organizada da seguinte forma: no Capítulo 2 são apresentados os fundamentos teóricos necessários para o entendimento dos métodos adotados; no Capítulo 3 é apresentada uma síntese dos trabalhos relacionados, expondo vantagens e desvantagens dos métodos existentes. A abordagem proposta é descrita no Capítulo 4. Por fim, no Capítulo 5 e 6, apresentamos os resultados e possíveis trabalhos futuros.

2 Fundamentos

Neste capítulo serão apresentados os conceitos fundamentais para o entendimento e desenvolvimento do trabalho. O arcabouço teórico foi dividido em quatro seções. A Seção 2.1 expõe as principais características de relatórios de defeitos, bem como os desafios enfrentados durante o seu processamento. Na Seção 2.2 apresentamos um comparativo entre os modelos neurais aplicados a diferentes entradas como textuais e categóricas. Na Seção 2.3, discutimos sobre as principais técnicas de pré-processamento para atributos textuais e categóricos. Por fim, na Seção 2.4, são descritas as principais métricas utilizadas para avaliar a tarefa de detectar duplicatas.

2.1 Relatórios de Defeitos

O processo de triagem de defeitos através de relatórios é feito em sistemas de rastreamento de problemas (SRP), como Bugzilla[1] e Atlassian[2]. Além disso, são sistemas que permitem qualquer usuário registrar um defeito sobre um produto de *software*, descrevendo através de um formulário o problema ocorrido. Geralmente, preenche-se um campo para título do problema, e outro campo para a descrição completa do ocorrido e outro para informações sobre o componente onde ocorreu a falha, além de ser feito o registro da data do ocorrido ou versão do *software*. Ao final, um novo documento é criado no sistema registrando uma nova falha (SUN et al., 2011; ZHANG et al., 2016).

Um SRP pode ser usado em qualquer projeto de *software*. No mercado de *software*, o Bugzilla é usado em mais de 136 companhias e organizações para gerenciar defeitos (LI et al., 2018). No entanto, existem SRPs em domínios industriais privados, cujas bases de dados não são públicas (LEE et al., 2017). Na prática, um SRP é um repositório centralizado de defeitos que pode ser usado por empresas para gerenciar problemas de *software*. Com isso, os relatórios são separados por projeto e organizados em domínios separados ou não. A

[1] https://www.bugzilla.org/
[2] https://br.atlassian.com/

Mozilla é uma empresa com projeto para o navegador Firefox[3] e projetos para o produto de e-mail Thunderbird[4]. O primeiro trata de navegação na web e o outro de serviços de e-mail. Nesse caso, os produtos são da mesma empresa, apesar de os domínios serem diferentes. Sendo assim, os relatórios desses projetos possuem vocabulários e jargões específicos, além de outros desafios que serão descritos a seguir.

2.1.1 Características de um Relatório de Defeito e as Duplicatas

Um relatório de defeito pode conter informações textuais e não textuais, preenchidas em dois momentos: inicialmente, durante a criação do relatório, e posteriormente, durante o processo de triagem, como explicam Uddin et al. (2017). Neste trabalho, destacamos cinco informações em um relatório de defeito: título, campo predefinido, anexo, descrição e comentário. Os campos são os mesmos usados por Zhang et al. (2016). Na Figura 3 pode ser visualizado um exemplo de relatório de defeito, em que podem ser identificadas posições de cada campo no documento. Informações textuais são todos os campos de texto, como título, descrição, anexo e comentários, enquanto as informações não textuais são todos os campos numéricos ou categóricos nos campos predefinidos do documento, como produto, componente, prioridade, severidade, resolução, status e versão. (RUNESON; ALEXANDERSSON; NYHOLM, 2007; UDDIN et al., 2017).

[3] https://www.mozilla.org/pt-BR/
[4] https://www.thunderbird.net/pt-BR/

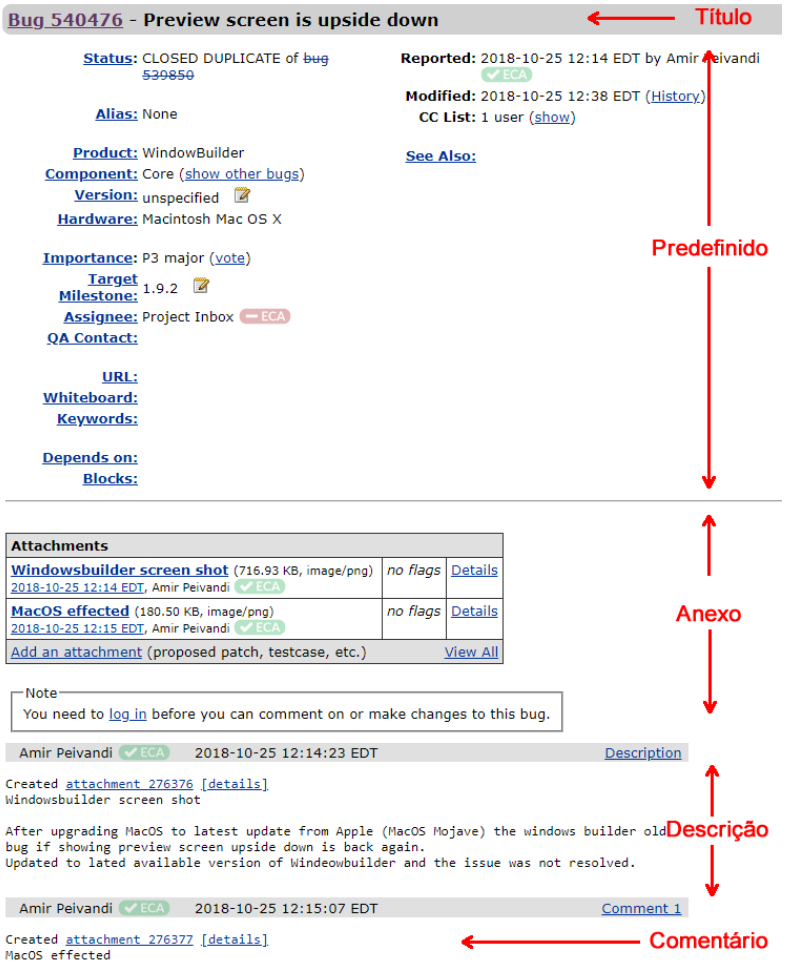


Figura 3 – Um exemplo de relatório de defeito no Eclipse.

Fonte: Próprio Autor

Durante a registro de um defeito, o título, o campo predefinido, o anexo e a descrição, geralmente, são as primeiras informações cadastradas. Em seguida, o defeito passa por uma triagem do especialista, podendo sofrer alterações nos campos predefinidos e inserção de comentários, que podem informar se trata-se de duplicação ou não de problemas. Nessa última informação, a rotulação, DUPLICADO ou *DUPLICATE* é aplicada sempre que o especialista identifica que dois ou mais relatórios estão descrevendo o mesmo problema. Assim, o especialista atualiza os respectivos relatórios por meio dos campos *status* e comentários.

No exemplo da Figura 3, temos um relatório de defeito identificado como #540476, reportado dia 25 de outubro de 2018, a respeito de um problema na tela do Eclipse no sistema operacional Mac OS X. Na parte inferior da figura, na seção dos comentários, um usuário informa que esse problema já

foi relatado anteriormente. No entanto, cabe a um especialista avaliar essa referência anterior e decidir o que fazer com o novo defeito. Um relatório novo, pode ser relacionado a um problema antigo que já foi resolvido ou a um problema ainda pendente de solução, reportado por outro usuário. Nesse caso, passamos a ter duplicatas quando, além da referência duplicata inserida pelo especialista, há um ou mais relatórios que descrevem o mesmo problema. Diante disso, somente um especialista pode identificar esse cenário e eleger um representante para um determinado grupo de duplicatas, agrupando as duplicatas pelo relatório mestre. Na Figura 1 do Capítulo 1 apresentamos um exemplo dessa estrutura, em que o relatório #6325 foi referenciado por muitos relatórios como duplicado e os referenciados foram apontados por outros relatórios, formando uma árvore.

Na Figura 3, o **título** é uma descrição curta sobre o defeito. Os campos denominados como **predefinido** agrupam informações sobre versão, componente e modelo do produto de *software*, além de informações sobre o problema como prioridade, estado da triagem e o relator do problema. A área do **anexo** é destinada ao código-fonte, imagens, vídeos ou qualquer outra fonte extra para auxiliar a triagem do defeito. O campo **descrição** corresponde à história completa do defeito, descrevendo os passos para reproduzir o problema e detalhes que não foram adicionados no título. O último campo, destacado como **comentário**, é onde os especialistas do time de desenvolvimento comentam sobre o problema (ZHANG et al., 2016; UDDIN et al., 2017).

2.2 Redes Neurais e Aprendizagem Profunda

As redes neurais são modelos de aprendizagem criados em meados da década de 1940 que fazem parte da família de algoritmos em aprendizagem de máquina (AM). São modelos usados em problemas como classificação ou regressão, ou como codificadores de informação, e sua arquitetura é composta por componentes como: entradas, camadas ocultas, representação latente com *embedding*, e uma função de perda para orientar o aprendizado. A entrada para esses modelos pode ser de qualquer natureza, como texto, imagem, áudio, ou video. Dessa maneira, as entradas são processadas por camadas subjacentes, denominadas

como camadas ocultas da rede. Nessas camadas, as entradas são processadas, aprendendo a representar as entradas em um novo espaço latente (os *embeddings*), a partir da combinação entre a entrada e pesos. Assim, o processo de aprendizado é feito em todas as camadas da rede, buscando encontrar um espaço de características latente que melhor descrevam as entradas, além de pesos que combinem com a entrada e resultem em um espaço latente melhor discriminado. Esse aprendizado, é orientado por meio da função de perda que minimiza o erro entre o valor previsto e o valor real, produzindo um modelo que sabe representar uma determinada entrada em um novo espaço latente (MASSON; WANG, 1990; GOODFELLOW et al., 2016).

Aprendizagem profunda (AP) é uma subárea de AM e visa estudar complexas combinações de modelos de redes neurais, misturando viéses de aprendizado e complexidade, além de formar arquiteturas heterogêneas que podem processar diferentes entradas e construir diferentes saídas (VEEN, 2016). O aprendizado profundo pode construir um poderoso codificador de informações, combinando diferentes padrões de aprendizado através de múltiplas entradas, além de construir conhecimento semântico sobre as entradas. O trabalho de Vinyals et al. (2015) demonstra como combinar entradas de texto e imagem para produzir legendas textuais em imagens. Além disso, muitos pesquisadores investigam essa linha pesquisa, processando entradas híbridas ao mesmo tempo (JING; TIAN, 2020). Na esfera de relatórios duplicados, Budhiraja et al. (2018a) separaram o conteúdo textual em duas entradas, título e descrição do relatório, concatenando os dois textos para um mesmo modelo. De outro modo, Deshmukh et al. (2017) demonstraram que utilizar tanto texto (título e descrição) em conjunto com os atributos categóricos (produto, componente, prioridade, severidade, resolução, *status*, e versão) traz um resultado melhor para a tarefa de detecção de duplicatas, codificando múltiplas entradas de um mesmo relatório de defeito para extrair diferentes conhecimentos semânticos de cada campo. A hipótese dos autores considera que modelos especializados para a natureza específica de cada campo leva à construção de representações melhores para a tarefa de detecção.

Nesse contexto, arquiteturas profundas possuem camadas ocultas responsáveis por combinar padrões e repassar o conhecimento adquirido para

camadas vizinhas. Assim, a função de perda pode ser aplicada para minimizar o erro da previsão gerado por cada camada da rede. Desta forma, modelar uma abordagem baseada em aprendizado profundo que possa receber as entradas textuais e categóricas sobre um mesmo relatório de defeito, possibilita ao modelo aprender características latentes sobre o documento que auxiliem na tomada de decisão para detectar duplicatas. Logo, campos textuais como título e descrição comportam uma estrutura sequencial entre o conjunto de palavras, além de existir sentenças compostas por vocabulários variados e variação semântica. Portanto, utilizar um modelo de aprendizagem profundo para representar textos possui a pré-condição de (1) considerar a sintaxe e coocorrência das palavras e (2) estabelecer o contexto das palavras. Além disso, textos em relatórios de defeitos podem conter conteúdo não natural gerado a partir de máquinas, como código-fonte, *logs* e *stack traces*, textos que seguem uma rígida sintaxe na sua estrutura, adicionando uma complexidade na informação dos relatórios, além de formar uma mistura de contextos.

Para um modelo de aprendizagem profunda representar campos não textuais, o mesmo precisa representar os atributos com base na correlação das categorias, identificando ausência e presença da informação para extrair padrões. No entanto, esses campos, costumam ser esparsos devido a grande quantidade de categorias presente em relatórios de defeitos.

Portanto, visando esclarecer as diferenças entre distintas redes neurais destinadas ao processamento textual e não textual de relatórios de defeitos, apresentamos nas próximas seções as escolhas adotadas neste trabalho para suprir as principais necessidades no desafio de processar campos de relatórios. Na Seção 2.2.1, descrevemos as redes de múltiplas camadas de perceptrons para processar informações textuais e não textuais, para então, nas Seções 2.2.2, 2.2.3, e 2.2.4, apresentamos as diferenças entre redes recorrentes, redes de convolução e representações de codificador bidirecional com transformações para texto. Em seguida, definimos a importância do aprendizado baseado na abordagem siamesa de redes neurais para comparar dois ou mais relatórios na Seção 2.2.5. Por fim, na Seção 2.2.6 apresentamos as características sobre funções de perda usadas em redes neurais.

2.2.1 MLP: Redes de Perceptrons em Multicamadas

Redes de *Perceptrons* em multicamadas, do inglês *Multilayer Perceptron* (MLP), são as redes neurais com a estrutura mais simples, composta por neurônios densamente conectados, como ilustra a Figura 4. Ao processar qualquer entrada x, esse modelo extrai o conhecimento semântico como saída, uma matriz Y composta pela combinação de cada entrada x_i com um peso w_i e qualquer viés b indicado por $Y = x_i\ w_i + b$, onde i representa um contador de 1 a n. Em trabalhos que envolvem a detecção de relatórios duplicados utilizando sequências textuais, cada entrada x_i geralmente são *tokens* de palavras ou distribuições de probabilidade de tópicos, ou campos categóricos de um relatório de defeito, transformados em vetores de codificação *one-hot*, uma representação binária para campos categóricos. Com algumas camadas MLP é possível extrair relacionamentos semânticos sobre a entrada de dados. Na saída dessa rede temos um vetor latente de tamanho D que descreve o conhecimento adquirido, permitindo aplicar tarefas de classificação, codificação e agrupamento (GOODFELLOW et al., 2016). Uma rede desse modelo pode ser usada para detectar se um determinado relatório de defeito é de fato um defeito ou não, com base no conteúdo textual, similar ao trabalho de Terdchanakul et al. (2017). Além disso, é possível utilizá-la como um codificador sobre relatórios de defeitos, para detecção de duplicatas (BUDHIRAJA et al., 2018a). Na abordagem de Budhiraja et al. (2018a), duas MLP são emparelhadas para processar o conteúdo textual de dois relatórios, codificando duas saídas que permitem identificar se os dois relatórios representam uma duplicata.

2.2.2 RNN: Redes Recorrentes

Para processamento textual, as redes recorrentes possuem uma infinidade de variantes, cobrindo basicamente o objetivo de processar sequências e listas, por meio da atribuição de importância ao passado das informações (BISHOP, 2006). Nesse sentido, processar texto de um relatório de defeito torna-se uma tarefa trivial, pois, uma vez que temos o aprendizado de características sequenciais, estamos considerando a coocorrência das palavras, memorizando padrões e, ao mesmo tempo, extraindo conhecimento semântico sobre as entradas. Portanto,

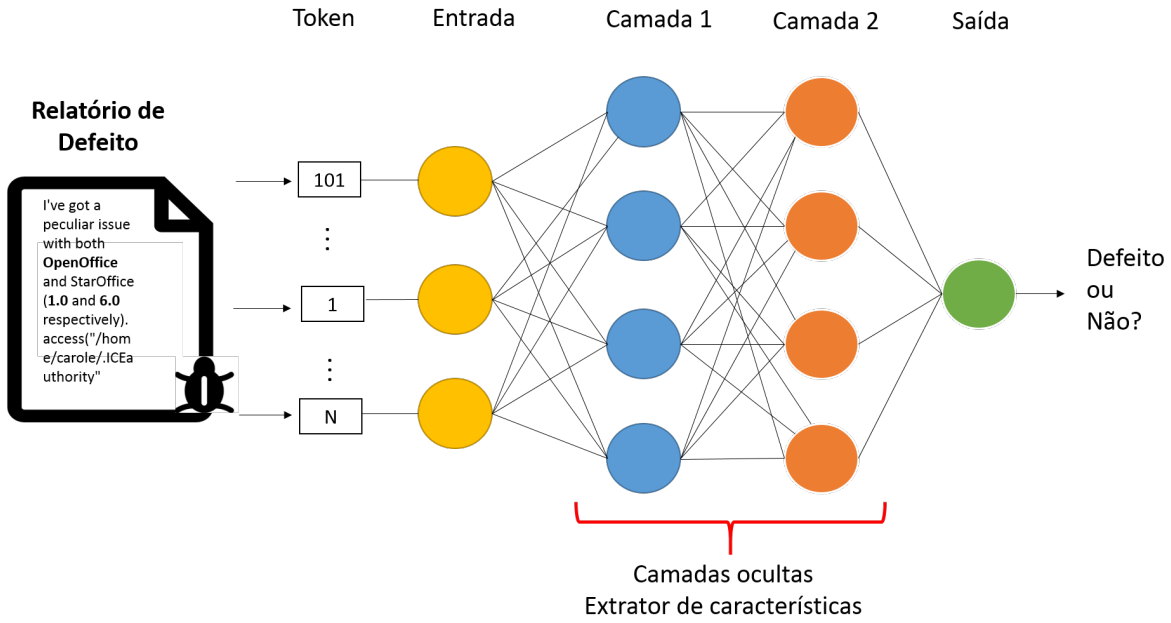


Figura 4 – Uma MLP com 2 camadas ocultas para classificar se um determinado relatório corresponde a um defeito ou não.

Fonte: Próprio Autor

como pré-requisito para processamento textual, começaremos discutindo sobre três variantes geralmente usadas para essa finalidade: a RNN, LSTM e BiLSTM. Esses modelos, são um grupo de redes com conceitos relacionados, que atuam sobre textos de maneiras distintas. Além disso, foram usadas por Deshmukh et al. (2017) em sua abordagem para detectar duplicatas.

As redes neurais recorrentes, denominadas em inglês como *Recurrent Neural Network* (RNN), correspondem a um modelo com neurônios encadeados, em que a saída de um neurônio é dada como entrada para outro, combinando a saída anterior com uma nova entrada em um tempo t_i. Dessa forma, é criada uma espécie de memória temporária no desdobramento do tempo. Com essa rede, pode ser criada uma saída temporária em cada tempo t_i, processando padrões de aprendizado com memória em cada instante, dando importância ao passado mais recente $t_i - 1$. Na Figura 5, apresentamos uma RNN em seu desdobramento temporal recebendo várias entradas. As RNN são conhecidas como recorrentes devido ao fato da mesma computação é operada para cada tempo t_i usando a saída da computação anterior, como demonstra a Equação 2.1,

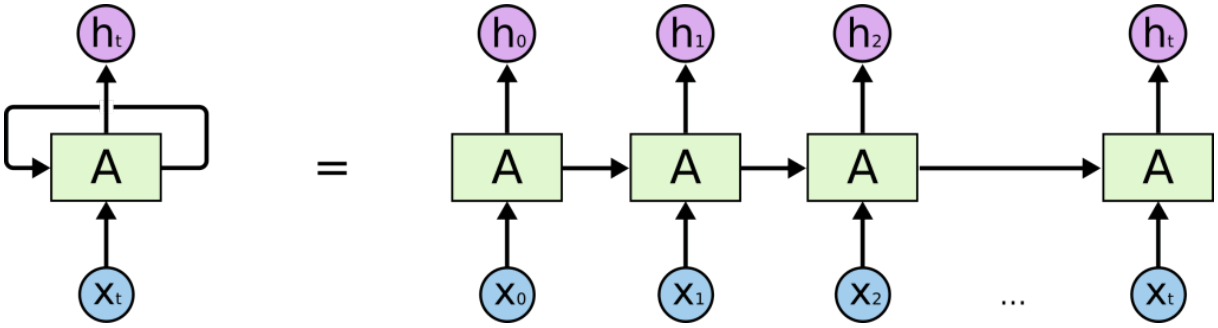


Figura 5 – Representação do desdobramento no tempo de uma RNN.

Fonte: *Ebook* de Data Science Academy (2019)

$$RNN(t_i) = f(W * X_{ti} + U * RNN(t_i - 1)) \tag{2.1}$$

W e U são os parâmetros do modelo e f é qualquer função não linear. $RNN(t_i)$ é a saída no ésimo tempo, que pode ser utilizado como está, ou pode ser alimentado novamente por uma construção parametrizada como *Softmax* (BISHOP, 2006).

Apesar de as RNN serem aplicadas em sequências, Deshmukh et al. (2017) reconhecem que esses modelos são limitados a sequências curtas e lentos no treinamento, além de serem suscetíveis a explosão de gradientes como explicam Pascanu, Mikolov e Bengio (2013). Assim, uma alternativa é a *Long Short-Term Memory* (LSTM), uma variante que possui mecanismos de memória e decisão, dividindo sua decisão através de três portões responsáveis por armazenar, esquecer e repassar uma informação, como demonstra a Figura 6. Na estrutura interna da LSTM, o aprendizado é adaptado para longas sequências, devido ao poder de decisão e memorização ao longo dos neurônios. Portanto, com as LSTM é possível extrair complexos padrões sobre as entradas e decidir quais armazenar e quais repassar (FAYYAZ et al., 2016). Em trabalhos relacionados, Qin e Sun (2018) usaram uma LSTM para classificar quando um relatório é um defeito. Em contexto de duplicatas, Deshmukh et al. (2017) demonstraram que as LSTM podem ser usadas para codificar atributos textuais de relatórios de defeitos. Construindo dessa maneira, representações latentes sobre os textos de modo a compará-los e descobrir quais documentos são semelhanças vetorialmente.

Ainda nesse contexto, visando enriquecer a entrada para redes neurais

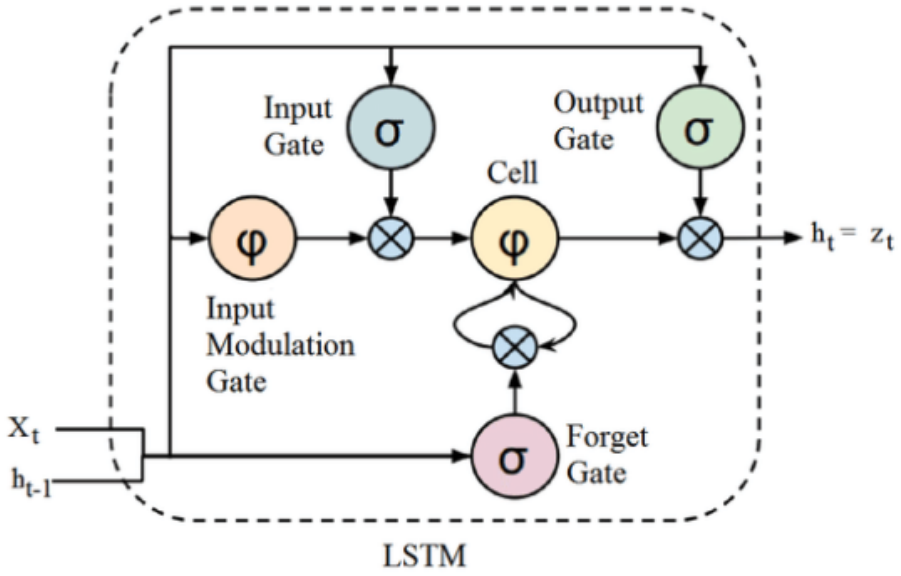


Figura 6 – Representação dos componentes de uma LSTM.

Fonte: Fayyaz et al. (2016)

e permitir a extração de contextos com representações mais significativas, Schuster e Paliwal (1997) propuseram a *Bi-directional Long Short-Term Memory* (bi-LSTM), com uma mudança de objetivo na LSTM original, capaz de processar qualquer sequência em duas direções. Nessa estratégia, dada uma entrada x, com tamanho n, é feito um processamento da mesma de frente para trás $\{x_1, x_{n-1}, .., x_n\}$ e de trás para frente $\{x_n, x_{n-1}, .., x_1\}$, usando o mesmo número de LSTM em duas camadas (TODI; MISHRA; SHARMA, 2018). Segundo Salehinejad et al. (2017), acessar conhecimento sobre vocabulários no futuro e passado a partir do estado atual aprimora o desempenho do resultado das redes neurais recorrentes. Mani et al. (2019) utilizaram esse modelo de rede para codificar relatórios de defeitos, mapeando a relação entre um defeito e um conjunto de desenvolvedores aptos a resolvê-lo, um problema de classificação. Na Figura 7, apresentamos um exemplo da arquitetura interna da BiLSTM.

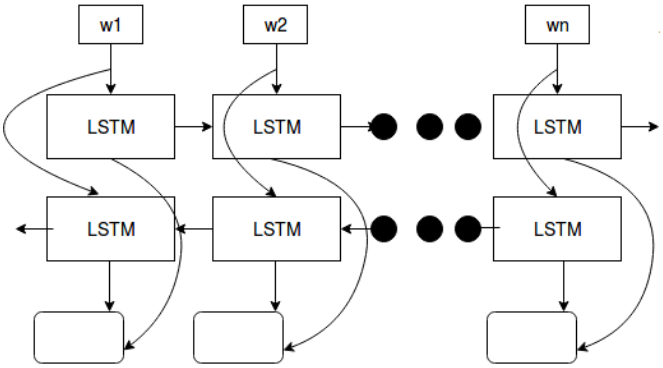


Figura 7 – Representação do aprendizado em uma bi-LSTM.

Fonte: Todi et al. (2018)

As redes temporais são eficientes para processamento de sequências, ao que correspondem os campos de relatórios de defeitos como, por exemplo, o título, que geralmente é descrito em uma sequência textual curta. Entretanto, para a descrição de relatórios de defeitos, esses modelos possuem uma limitação diante de longas sequências, porque o aumento do número de neurônios aumenta o número de parâmetros, tornando o modelo complexo. Além disso, as RNN apresentaram uma alta dependência entre as entradas, visto que a entrada de um neurônio é a saída de outro, tornando o treinamento muito lento. Por outro lado, existem outras abordagens para processamento de longas sequências, que incluem as redes neurais convolutivas, descritas a seguir.

2.2.3 CNN: Redes de Convolução

Como mencionado, a descrição de um relatório de defeito é muito mais extensa do que o título, inviabilizando o uso de RNNs, devido à complexidade desses modelos e o tempo de treinamento (BISHOP, 2006). Por essa razão, faz-se necessário a substituição do aprendizado por uma rede neural que processe longas sequências textuais. Nesse sentido, as redes de convoluções podem ser uma alternativa. Do Inglês *Convolutional Neural Network* (CNN), as redes neurais convolutivas são modelos que operam com partes diferentes da entrada, testando diferentes visões ou filtros, sendo muito eficientes como extratores de características e processamento de sequências textuais (SANTOS; GATTI, 2014). A CNN, assim como a RNN, possui variantes, como, por exemplo, as, CNN dilatadas, modelos capazes de aumentar o campo de visão do aprendizado,

sem a necessidade de aumentar exponencialmente o número de parâmetros do modelo (YU; KOLTUN, 2015).

O processo de aplicar padrões sobre entradas é chamado de convolução, onde uma CNN precisa ter um número de operações definido, o filtro (KIM, 2014). Esse filtro, guia o número de vezes que uma CNN opera sobre as entradas. Assim, a rede neural constrói mapas de atributos combinando com a entrada para obter padrões diferentes sobre a entrada. No final, a rede combina os aprendizados gerados em uma única saída de tamanho fixo. Para então, utilizar a saída como entrada para outra rede como uma MLP. Uma convolução pode ser definida como funções $f(x)$ e $g(x)$, onde resultamos a convolução, $h(x)$ = $f(x) * g(x)$, conforme a equação (2.2). A convolução no instante α é igual à área da interseção entre $f(x)$ e $g(\alpha - x)$ (GONZALEZ; WOODS, 2002).

$$h(x) = f \otimes g = \int_{-\infty}^{\infty} f(\alpha)g(x - \alpha)d(\alpha) \tag{2.2}$$

Na Figura 8, apresentamos uma CNN com filtro de tamanho 4, utilizada para um problema de análise de sentimento em textos. Assim, os padrões gerados pela rede através dos filtros são combinados em uma única saída, usando a técnica de combinação dos filtros chamada de *pooling*. No fim, o *pooling* irá combinar os filtros construídos pela rede calculando uma média ou valor máximo entre os mapas de atributos (KIM, 2014). No contexto de duplicatas, Deshmukh et al. (2017) utilizaram uma CNN com filtros de 3, 4 e 5 para codificar descrições de relatórios de defeitos. No entanto, apesar dos resultados demonstrarem um impacto significativo na detecção das duplicatas, esse modelo, assim como as RNN, não possuem mecanismos de atenção incorporados em sua arquitetura, desconsiderando a mudança de contexto para textos não naturais. Diante disso, outra estratégia será apresentada na seção 2.2.4, discutindo as vantagens em utilizar atenção textual e codificação bidirecional através de transformações.

Considerando trabalhos recentes, pode-se afirmar que as CNN são eficientes para processamento de sequências, resolvendo o problema de processamento de longas sequências enfrentado pelas RNN. Todavia, são modelos que não aprendem a posição e orientação das entradas (SABOUR, 2019; JADER-

BERG et al., 2015). Uma variante da CNN, é a rede neural dilatada, cujo objetivo é aumentar o campo receptivo dos filtros aplicados nas entradas, através de um fator de dilatação aplicado como $\{k = 1, k = 2, k = 3, ..., k = n\}$, melhor ilustrado na Figura 9 (YU; KOLTUN, 2015). Apesar de não resolver os problemas citados, essa estratégia amplia o processamento de longas sequências sem aumentar exponencialmente o número de parâmetros como acontece nas RNN.

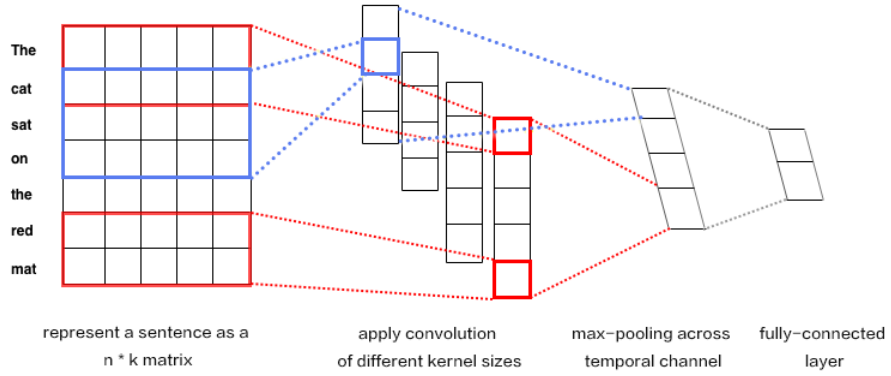


Figura 8 – Um exemplo da CNN com 4 filtros.

Fonte: Kim et al. (2014)

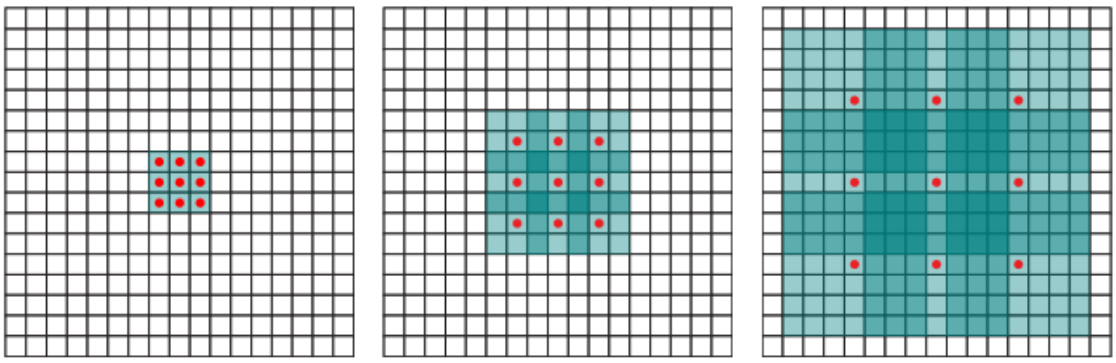


Figura 9 – Um exemplo da CNN dilatada com três fatores de dilatação. (a) fator k=1 campo receptivo igual a 3x3. (b) fator k=2 campo receptivo igual a 7x7. (c) fator k=3 campo receptivo igual a 15x15

Fonte: Yu e Koltun (2015)

2.2.4 BERT: Representações de Codificador Bidirecional com Transformadores

Mecanismos de atenção são uma maneira de ponderar os dados de entrada em uma rede neural com base em sua importância para uma determinada

tarefa (VASWANI et al., 2017). Pesquisadores como Deshmukh et al. (2017) citam a possibilidade de usar atenção em uma arquitetura personalizada, apesar de não apresentarem experimentos em sua abordagem. Nesse sentido, desde a introdução desses mecanismos por Bahdanau, Cho e Bengio (2014) e Luong, Pham e Manning (2015), problemas complexos envolvendo longas sequências de textos, como tradução automática, mostraram melhorias de desempenho. Assim, as soluções anteriores implantadas através de RNN e CNN começam a ser menos utilizadas como soluções, dando origem a novos conceitos como transformações, que são técnicas que processam sequências textuais transformando-as em outras sequências textuais, por isso o nome. Essa técnica foi introduzida por Vaswani et al. (2017), abrindo novas maneiras para processamento de sequências textuais.

As tarefas mais populares de linguagem natural tiveram melhorias de resultados usando técnicas de atenção, que focam em representar informações baseado em importância (CHOROWSKI et al., 2015; LUONG; PHAM; MANNING, 2015; TRAN; NIEDEREÉE, 2018; YIN et al., 2016). Como resultado, foi introduzido o modelo *Bidirectional Encoder Representations from Transformers* (BERT) (JAIN; WALLACE, 2019), usado em processamento de linguagem natural para uso geral, em tarefas como resposta a perguntas, inferência de linguagem natural e detecção de paráfrase (LAN; XU, 2018; YOUNG et al., 2018). Assim, temos um aprendizado que extrai relações contextuais em um texto (DEVLIN et al., 2018).

O núcleo da BERT é formado por vetores latentes com recursos ponderados, onde, dada uma sequência X de tamanho n, as palavras $\{x_1, x_2, .. \text{ e } x_n\}$, são identificadas por um vetor contínuo, o *embedding*, que nesse caso, estará representando o vocabulário de palavras com tamanho m, representando o espaço latente para cada palavra. Assim, podemos esboçar em alto nível a saída $attetion$ construída pela BERT, dada pela matriz n por m, ponderada e transformada através de uma função posicional (f). Para cada linha na matriz $attention$, um novo vetor de tamanho m é calculado a partir de uma média de todas as entradas X ponderadas com pesos w (equação 2.3). Portanto, visando facilitar o entendimento desse mecanismo, na Figura 10 ilustramos um exemplo para a frase "The file came", onde representamos cada palavra como um

embedding de palavras, e ao lado produzimos um vetor posicional ponderado com pesos para cada palavra. Dessa forma, o modelo relaciona as palavras entre si, aplicando um sistema de pesos que permite selecionar informações relevantes durante o aprendizado, estabelecendo contextos entre as palavras, o que segundo Jain e Wallace (2019) e Devlin et al. (2018), esse método fortalece a compressão do texto para o modelo. Nesse sentido, de posse do conhecimento da relação entre as palavras, é possível prever a próxima palavra de acordo com a coocorrência delas.

$$attention_{[n,m]} = \frac{1}{n}\sum_{i=1}^{n} f(X_i).w \tag{2.3}$$

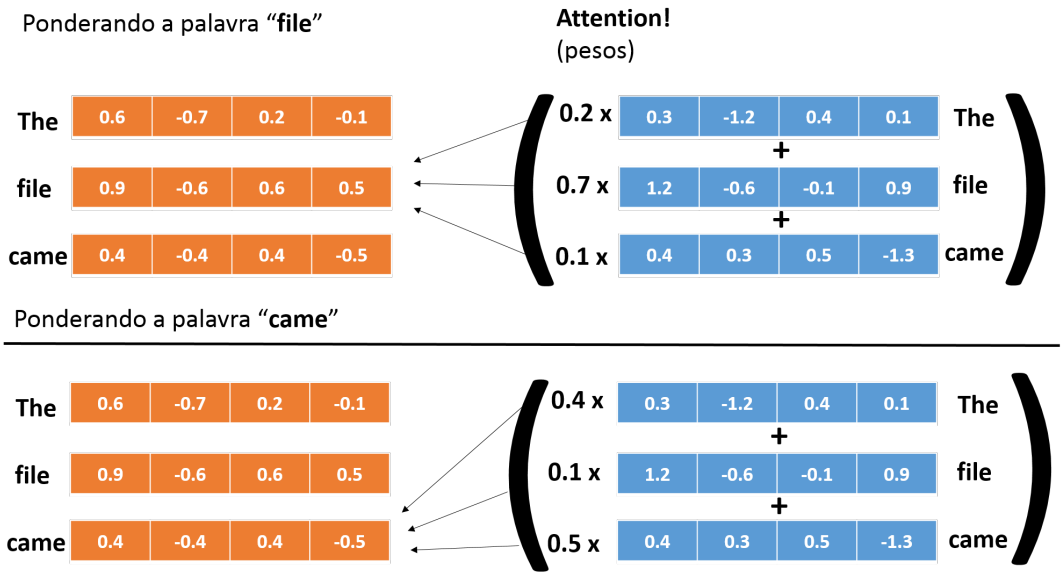


Figura 10 – Ilustração do mecanismo de atenção aplicado na frase "The file came".

Fonte: Próprio Autor

Na Figura 11 apresentamos uma comparação entre a BERT e outros modelos, onde podemos visualizar que a BERT relaciona todas as palavras, de maneira bidirecional, uma característica que Devlin et al. (2018) destacam como a principal diferença do modelo em relação a outros como RNN e CNN,

além do *OpenAI GPT* (VIG, 2019b) e *ELMo* (PETERS et al., 2018), que são propostas anteriores a BERT. Nesse sentido, percebemos que o aprendizado na *OpenAI GPT* relaciona as palavras da esquerda para a direita, de forma unidirecional, enquanto no *ELMo*, o aprendizado acontece baseado em recursos, semelhante a RNN e CNN. Por essa razão, a BERT revelou-se como o melhor modelo para representação textual, devido ao desempenho ter sido aprimorado ao considerar a relação entre todas as palavras do texto através de atenção, e consequentemente, construir representações com uma compreensão do contexto das palavras.

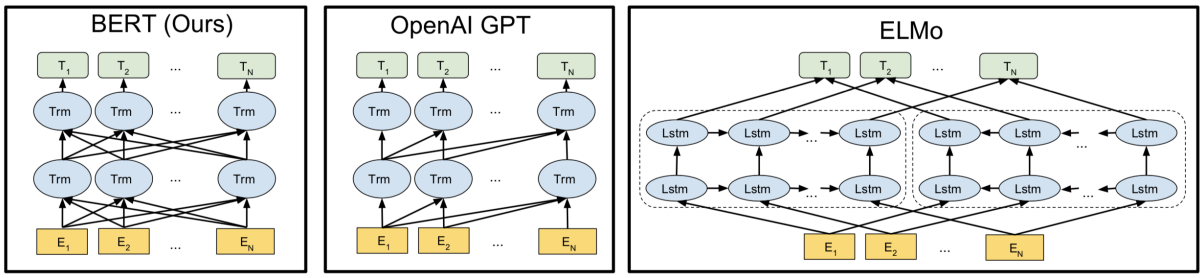


Figura 11 – Arquitetura BERT em comparação com a OpenAI GPT e ELMo.

Fonte: Devlin et al. (2018)

Como mencionado, a BERT pode ser usada em diferentes cenários para processamento natural de linguagem. Com isso, podemos representar as características textuais sobre relatórios de defeitos e gerar representações latentes sobre o texto. Nesse sentido, a BERT possui algumas vantagens como o pré-treino e o suporte a multilinguagem. O pré-treino é a prática de utilizar um modelo previamente treinado corrigindo o problema de partida a frio ou *cold start* de modelos baseados em aprendizagem de máquina (SINGH et al., 2019; VIRTANEN et al., 2019). O suporte a multilinguagem, consiste em um modelo capaz de processar qualquer idioma de linguagem natural. Dessa maneira, temos um modelo que se beneficia sobre corpus abundantes de palavras, e capaz de construir ricas representações textuais (DEVLIN et al., 2018). Outra aplicação da BERT, é a possibilidade de treinar o modelo em um novo vocabulário de palavras para aprender a representar vocabulários específicos, como acontece no vocabulário de relatórios de defeitos.

Segundo Devlin et al. (2018) existem dois modelos prontos para uso,

BERT_{BASE} e BERT_{LARGE}. O primeiro, com 110 milhões de parâmetros, 12 blocos de transformações, representações textuais de tamanho 768 e 12 camadas de atenção. O segundo, 340 milhões de parâmetros, 24 blocos de transformações, representações textuais de tamanho 1024 e 16 camadas de atenção. Dessa maneira, nossa pesquisa pode ser caracterizada como um problema semelhante à detecção de paráfrase, uma vez que desejamos detectar quando dois textos (juntamente com suas características categóricas) de relatórios duplicados estão descrevendo o mesmo assunto. Portanto, podemos usar um dos modelos pré-treinado para codificar o conteúdo textual de relatórios de defeitos, e desfrutando de suas representações textuais para identificar quais relatórios possuem semelhanças vetoriais. Apesar de nenhum trabalho envolvendo duplicatas utilizar BERT, Zafar, Malik e Walia (2019), aplicaram em engenharia de *software* para extrair contexto das mensagens de *commits* de desenvolvedores, fornecendo uma rica representação textual sobre as mensagens.

2.2.5 Rede Neural Siamesa para Comparar Objetos

Para detectar relatórios duplicados usando aprendizagem profunda são necessárias redes neurais capazes de comparar objetos através de similaridade vetorial, ou seja, dado um conjunto de objetos, a rede possa compará-los para determinar similaridade. Na literatura, os problemas como detecção de paráfrase e identificação de perguntas e respostas são conhecidos por adotarem redes neurais siamesas (LAN; XU, 2018; YOUNG et al., 2018), uma arquitetura tipicamente modelada para comparar pares de objetos, que possui alta capacidade de generalizar representações sobre os objetos comparados.

Uma rede neural siamesa tipicamente é uma arquitetura originalmente modelada para receber duas entradas, que podem ser imagem, texto ou qualquer outro objeto. Entre a comunidade de processamento natural de linguagem, existem vários trabalhos que empregam predominantemente a arquitetura siamesa (RANASINGHE; ORASAN; MITKOV, 2019; REIMERS; GUREVYCH, 2019; IMANI et al., 2019). Nessa estratégia, a transformação nas entradas constrói vetores numéricos para representação semântica, e garante que os mesmos estejam num único espaço latente de características, por meio de um sistema

de compartilhamento de pesos w (BROMLEY et al., 1994). Nessa arquitetura, Bromley et al. (1994) demonstrou que duas redes ($Rede\ A$ e $Rede\ B$) podem trabalhar lado a lado para representar um par de objetos, com o objetivo de aproximar vetorialmente as duas entradas ($Entrada\ A$ e $Entrada\ B$), diminuindo a distância vetorial entre entradas similares e aumentando a distância para aquelas distintas, por meio de uma função de perda (CHOPRA; HADSELL; LECUN, 2005). Deshmukh et al. (2017) confirmaram que nessa arquitetura é possível inserir mais de duas entradas ao mesmo tempo.

A ideia central da rede neural siamesa é utilizar redes neurais conjuntas que compartilhem pesos w, e representações latentes de um mesmo espaço de latente de características, como ilustra a Figura 12. As redes A e B, adotadas nessa topologia podem ser qualquer outra arquitetura de rede neural, como Chopra et al. (2005) ao adotarem duas redes de convolução para detectar faces da mesma pessoa em um problema de reconhecimento de réplicas faciais.

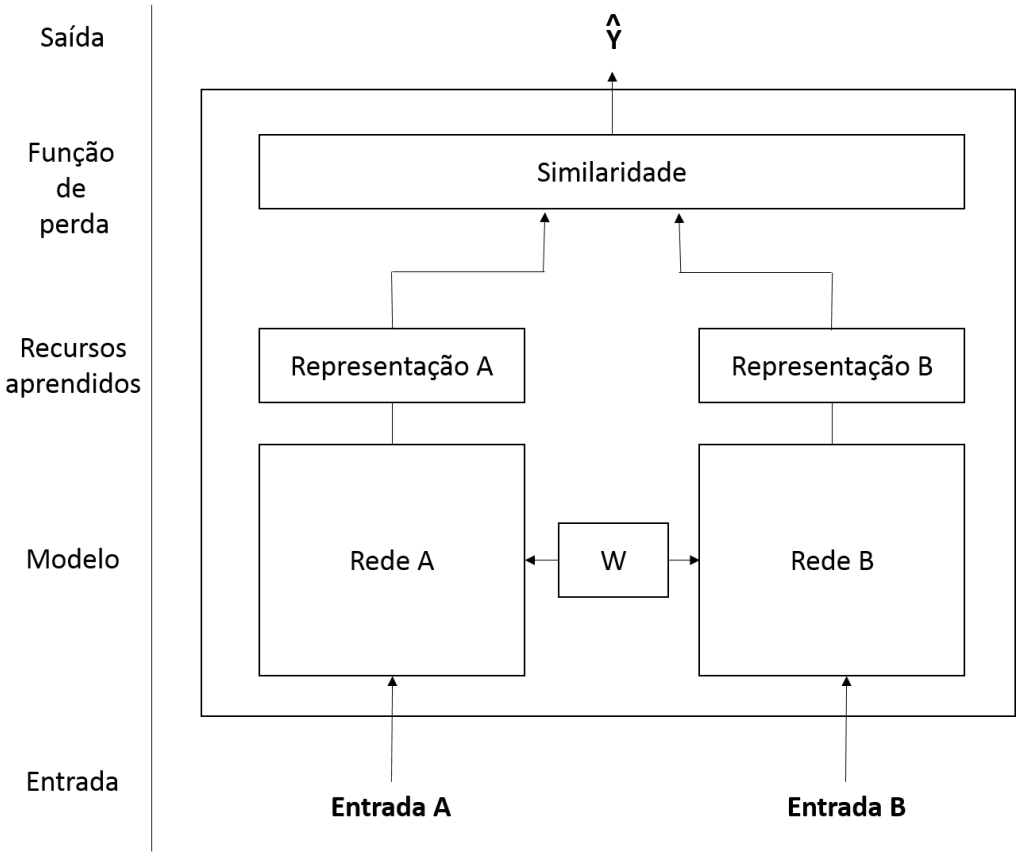


Figura 12 – Rede siamesa para comparar Objetos

Fonte: Próprio Autor

O poder dessa topologia está no uso do mesmo espaço de características construído para as representações latentes dos objetos comparados, tornando o modelo mais leve devido ao compartilhamento de pesos, e a produção de representações que possam ser usadas para visualização, agrupamento e outros fins (WANG; MI; ITTYCHERIAH, 2016). No contexto de relatórios duplicados, podemos treinar um modelo para receber x relatórios de defeitos para descobrir as semelhanças entre instâncias através das representações produzidas pelo modelo na camada destacada como "recursos aprendidos" na Figura 12. Vale destacar que o desempenho dessas arquiteturas dependem da função de perda utilizada para guiar o aprendizado, o que será apresentado na próxima seção.

2.2.6 Função de Perda em Redes Neurais

Uma função de perda é uma medida para avaliar quão bem um modelo neural aprendeu, fornecendo instruções sobre como melhorá-lo, calculada pela diferença entre a previsão de saída Y_{pred} e o valor verdadeiro Y_{true} (REIMERS; GUREVYCH, 2019; NIELSEN, 2015). Na literatura, podemos dividir funções de perda em três categorias: *perda baseada em classificação, perda baseada em regressão e perda baseada em terceto*. As estratégias possuem saídas diferentes, além de dependerem da entrada que precisa ser processada. Portanto, adotar qualquer uma delas dependem exclusivamente do problema a ser resolvido.

- **Perda baseada em classificação**: prevê a saída de um conjunto de valores categóricos finitos. Um exemplo dessa estratégia é a função Softmax definida pela equação 2.4,

$$Y_{pred} = SOFTMAX(W(f(A) \cup f(B) \cup |f(A) - f(B)|)) \tag{2.4}$$

  onde A e B representam valores de saída e o esperado, $f(A)$ a função que representa a previsão e $f(B)$ a função que representa a saída esperada. Dessa maneira, as duas funções em conjunto com W pesos treináveis concatenam-se para tirar a diferença entre elementos $|f(A) - f(B)|$ e multiplicar por W, gerando a saída Y_{pred}. A estratégia *Softmax* é usada

geralmente em problemas com vários rótulos quando precisamos criar saídas de probabilidade para cada rótulo. Outra estratégia é utilizar a *Sigmoid* na saída da rede neural para calcular uma probabilidade única (problema binário). Sendo assim, usar a *Softmax* para problema binário, funciona da mesma forma que a *Sigmoid*, gerando em duas probabilidades a saída da rede. Em trabalhos relacionados, Budhiraja et al. (2018a) adotaram a função *Sigmoid* para comparar pares de relatórios duplicados, construindo probabilidades sobre as duplicatas.

Usando a *Sigmoid* e a *Softmax*, podemos calcular a função de perda entropia cruzada binária definida pela equação (2.5), onde Y é a classe e $p(y)$ a probabilidade da instância dada como entrada ser da classe verdadeira Y para todas as n instâncias (ROBERT, 2014; LIU et al., 2016; BRUCH et al., 2019). A função de entropia cruzada binária calcula a diferença entre duas distribuições de probabilidade, sobre um conjunto de dados: p o verdadeiro e q a estimativa. Desse modo, o objetivo da função é aproximar a verdadeira distribuição p com alguma distribuição q. Na prática, um classificador baseado nessa função deverá construir representações que melhor descrevam os relatórios de defeitos, aproximando a representação prevista q da distribuição verdadeira p.

$$H_p(q) = -\frac{1}{n}\sum_{i=1}^{n} Y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)) \tag{2.5}$$

- **Perda baseada em regressão**: É uma função aplicada para prever valores contínuos, diferentemente da perda baseada em classificação. Aqui, a semelhança de cosseno entre duas frases que incorporam A e B pode ser calculada e usar a função de perda do erro médio quadrático (MSE) como objetivo através da equação (2.6) e (2.7). Apesar de nenhum trabalho relacionado ter aplicado essa função, a estratégia pode aproximar relatórios de defeitos semelhantes por meio da construção de representações que minimizem a diferença vetorial entre a similaridade prevista e a real,

$$Y_{pred} = (Y_{true} - DIST\hat{A}NCIA(f(A), f(B)))^2 \tag{2.6}$$

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(Y_{true} - DISTÂNCIA(f(A), f(B)))^2 \tag{2.7}$$

nas equações, A e B são a incorporação da sentença, $f(A)$ e $f(B)$ as representações das sentenças e $DISTÂNCIA$ a função de similaridade.

- **Perda baseada no terceto:** É uma estratégia que usa três instâncias para treinamento. Um candidato aleatório, um exemplo positivo ao candidato aleatório, e um negativo. Nessa perda, temos como saída um intervalo contínuo entre $\{0, .., 1\}$ correspondente a similaridade entre as instâncias positivas e negativas, como demonstra a equação (2.8),

$$Y_{pred} = MAX(0, C + DISTÂNCIA(f(A), f(P)) - DISTÂNCIA(f(A), f(N))) \tag{2.8}$$

na equação, a constante C geralmente é definida como 1 e, aplicando uma margem de distância entre os valores de similaridade vetorial, esse valor pode ser configurado livremente. A instância A é o candidato chamado âncora, P é um exemplo positivo de A e N é uma instância negativa de A. Em todas as instâncias (A, P, N) são aplicadas funções f que geram a representação associada para cada instância. A função $DISTÂNCIA$ é uma medida de similaridade como a euclidiana ou cosseno. No trabalho de Deshmukh et al. (2017), o cosseno foi a medida de similaridade utilizada.

A função de perda associada a essa estratégia de perda baseada no terceto é a *Triplet Loss (TL)*, uma função que gera aprendizados para agrupar instâncias semelhantes no espaço latente, sem incluir as informações de centroides. Essa função maximiza a semelhança vetorial entre a representação $f(A)$ e $f(P)$, enquanto minimiza para $f(A)$ e $f(N)$. Em um cenário de detecção de duplicatas, podemos aproximar as duplicatas enquanto afastamos as não duplicatas, construindo grupos com instâncias duplicatas. Nesse sentido, ao usar uma arquitetura siamesa, normalmente usa-se a TL, devido à melhoria no desempenho dos modelos, e um aprendizado para aproximar entradas positivas semelhantes, enquanto afasta as negativas espacialmente, construindo assim, grupos de instâncias semelhantes.

Os trabalhos que adotaram essa função TL foram o de Deshmukh et al. (2017), na detecção de duplicatas, e do Schroff et al. (2015b) no problema de reconhecimento facial.

Como a TL é eficiente para agrupamento de instâncias semelhantes, acreditamos que o uso de centroides possa ser útil para aproximar rapidamente essas instâncias, além de construir grupos de duplicatas melhor organizados no espaço latente. No entanto, motivaremos com mais detalhes no Capítulo 4 sobre a importância dos centroides na função perda TL.

As funções de perda possuem um papel fundamental para guiar o aprendizado das redes neurais, dependendo exclusivamente do problema a ser resolvido, além de serem medidas sensíveis às entradas. No contexto de relatórios duplicados, a perda baseada em terceto é a estratégia que mais se adéqua as reais necessidades de processar instâncias duplicatas e não duplicatas, visto que Deshmukh et al. (2017) apresentaram resultados superiores em comparação ao trabalho de Budhiraja et al. (2018a), sendo que o segundo usaram a perda baseada em classificação. No Capítulo 4, demonstraremos nossa proposta de modificação algébrica na função original da TL para considerar informações coletivas sobre as duplicatas (centroides).

2.3 NLP: Processamento de Linguagem Natural

O treinamento de modelos neurais demanda que as informações de relatórios de defeitos sejam processadas para valores numéricos. Nesse sentido, descrevemos os processamentos empregados em campos textuais e categóricos que normalmente são utilizados pela comunidade NLP. Portanto, as técnicas apresentadas, são: *tokenização*, *embeddings de palavras*, *representação em tópicos de palavras* e *codificação one-hot*.

2.3.1 Tokenização: Representação de Palavras em Tokens

O termo tokenização corresponde ao processo que ocorre quando dado um determinado texto, divide-se uma sequência textual em palavras, transformando as palavras em tokens numéricos. Os tokens podem ser representados através

de um número único (id) ou a frequência da palavra. As palavras podem ser separadas por espaço, vírgula ou qualquer outro delimitador (STAVRIANOU; ANDRITSOS; NICOLOYANNIS, 2007).

A tokenização na BERT possui dois conceitos: *In-Vocabulary words (InV)* e *Out-of-Vocabulary (OOV)*. O primeiro refere-se às palavras conhecidas pelo modelo pré-treinado. O segundo refere-se às palavras não conhecidas pelo modelo; nesse caso, são adicionados caracteres '**##**' como prefixo dessas pseudopalavras (MULLER; SAGOT; SEDDAH, 2019). Além disso, tokens especiais são adicionados no início e no fim da sentença, "**[CLS]**" e "**[SEP]**", como sugerido por Devlin et al. (2018). Portanto, para tokenizar a frase "**Need workspace setting to control colors of LineNumberRulerColumn**", temos os seguintes passos de tokenização, como demonstra a Figura 13.

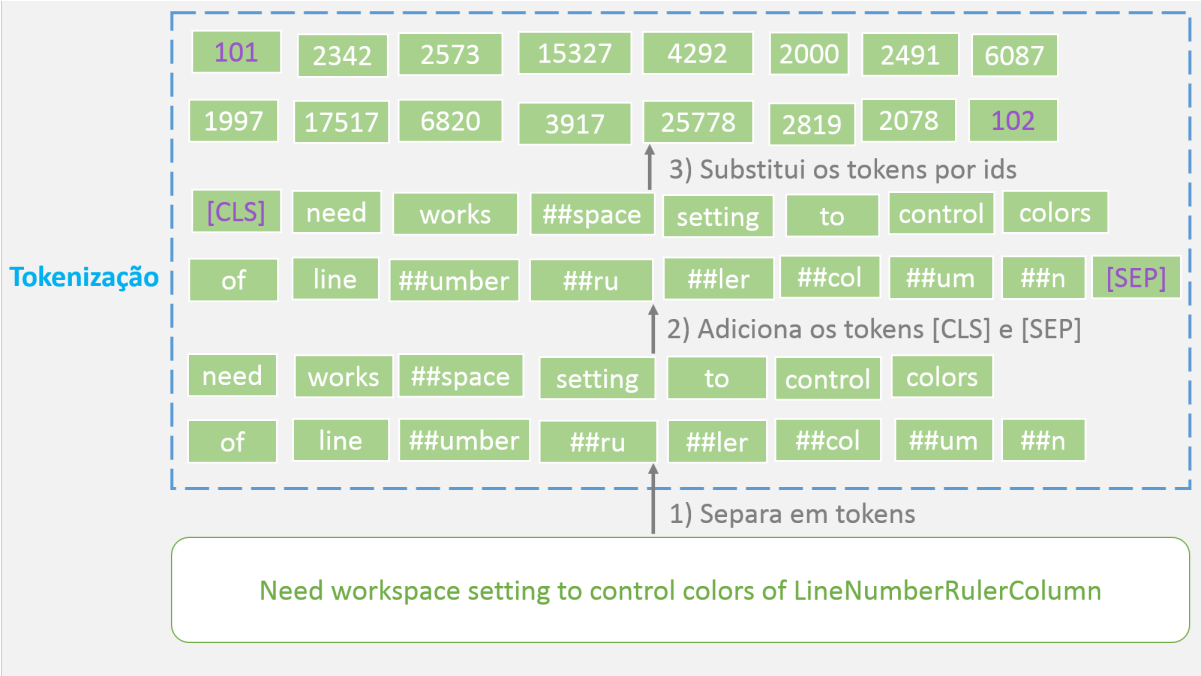


Figura 13 – Tokenização de uma sequência textual

Fonte: Próprio Autor

Analisando a Figura 13, mesmo para vocabulários desconhecidos, como "**workspace**" e "**LineNumberRulerColumn**", podem ser incluídos na tokenização por meio da estratégia OOV. Essas palavras, são frequentes nas três bases, como podemos ver no conjunto de palavras tokenizadas na Figura 14 e na Figura 16. Abaixo, listamos alguns exemplos OOV:

- **Eclipse**: (1) dialog = dial + ##og; (2) ident = id + ##ent; (3) runtime = run + ##time; (4) bench = ben + ##ch; (5) exception = ex + ##exception; (5) refresh = re + ##fresh;

- **NetBeans**: (1) screenshot = screen + ##hot; (2) runtime = run + ##time; (3) exception = ex + ##exception; (4) breakpoint = break + ##point; (5) debug = de + ##bug; (6) trace = tra + ##ce;

- **Open Office**: (1) dialog = dial + ##og; (2) delete = del + ##ete; (3) toolbar = tool + ##bar; (4) curso = cur + ##so; (6) formatting = format + ##ting; (5) hyperlink = hyper + ##link.

Nesse sentido, podemos repetir o processo de tokenização no título (*short_desc*) e descrição (*description*) dos relatórios de defeitos, normalizando as informações textuais do documento. Com essa representação, as redes neurais recebem uma compreensão de ordem e coocorrência das palavras, além de poderem incluir palavras novas.

Figura 14 – Conjunto de palavras na base Eclipse.

Fonte: Próprio Autor

Figura 15 – Conjunto de palavras na base NetBeans.

Fonte: Próprio Autor

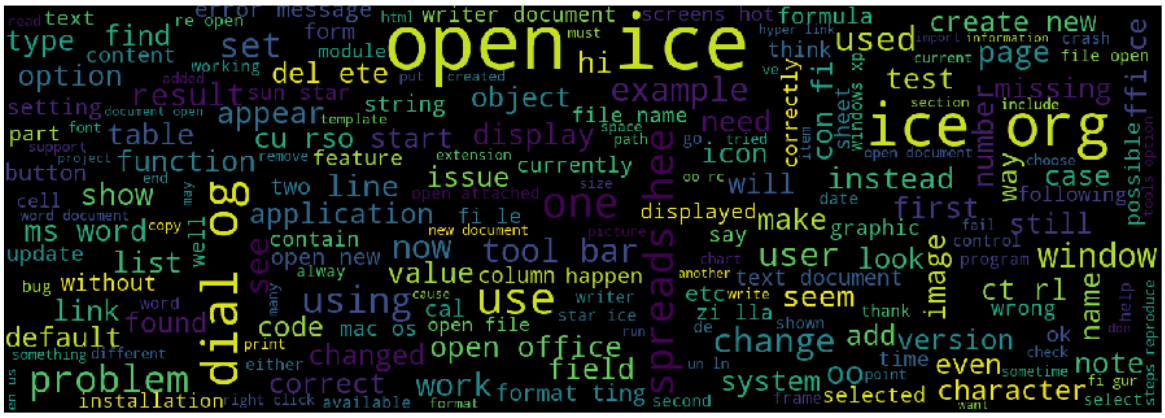


Figura 16 – Conjunto de palavras na base Open Office.

Fonte: Próprio Autor

2.3.2 LDA: Representação baseada em Alocação Latente de Dirichlet

Modelagem em tópicos é uma das técnicas mais conhecidas em mineração de texto, que extrai dados latentes e relações de contexto entre documentos textuais (VIRTANEN et al., 2019; XIA et al., 2019). Segundo Blei et al. (2003), *Latent Dirichlet Allocation (LDA)* é um modelo probabilístico generativo para coleções discretas como documentos de texto, que representa, por meio de um conjunto finito de probabilidades, tópicos de palavras. Formalmente, uma palavra é a unidade básica de um documento textual, a informação discreta. Um documento é definido como uma sequência de n palavras definidas como

V=$(v_1, v_2, ..., v_n)$, onde vn é a enésima palavra na sequência. Um *corpus* é visto como uma coleção de m documentos. No entanto, ao calcularmos distribuição de tópicos para corpus através do LDA, um documento passa a ser representado por um conjunto finito de probabilidades sobre tópicos e os tópicos passam a ser um conjunto finito de probabilidades sobre palavras. Na Figura 17, apresentamos a hierarquia adotada pelo LDA para representar tópicos.

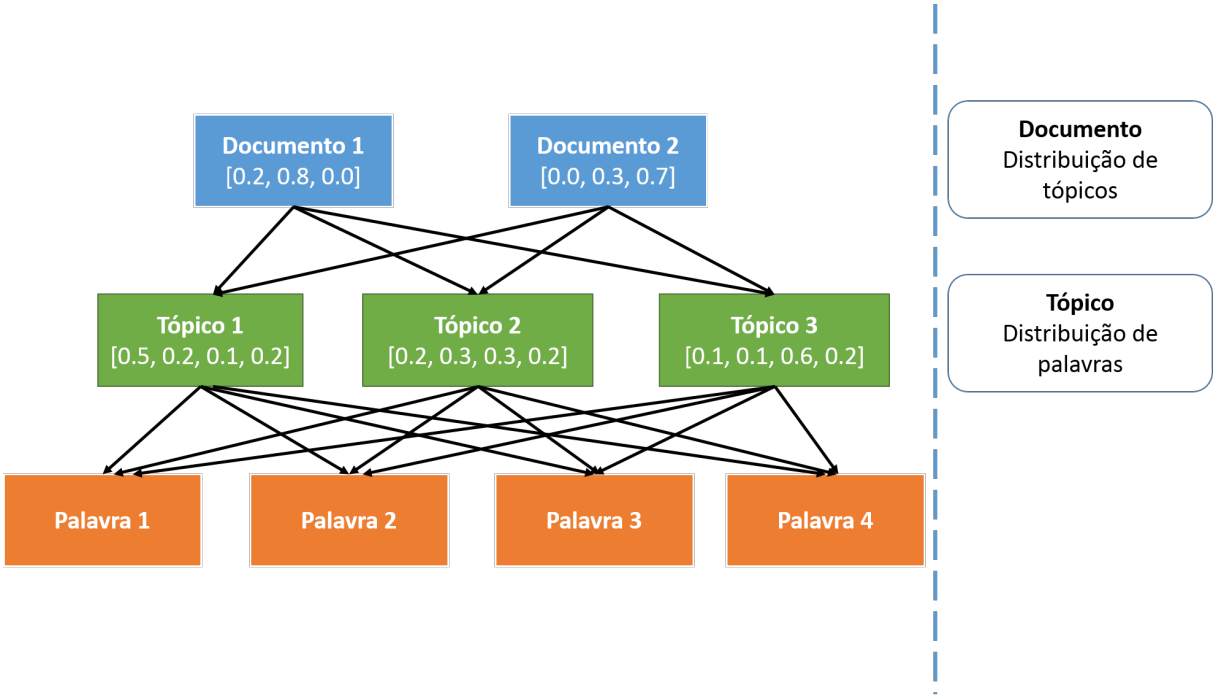


Figura 17 – Composição hierárquica das relações entre documentos, tópicos e palavras no LDA.

Fonte: Próprio Autor

Organizar as duplicatas através de tópicos, fornece uma ideia de contexto sobre os relatórios, permitindo julgar os grupos segundo sua distribuição de palavras. Na Figura 18, apresentamos a visualização em duas dimensões sobre 30 tópicos gerados nas bases do Open Office e Eclipse, usando o LDA, onde percebemos um espaço latente com uma certa organização, além de grupos que compartilham os mesmos tópicos. Esse número de 30 tópicos foi determinado através de uma busca exaustiva entre o melhor número de tópicos em relação ao menor valor de perplexidade para a base, mensurando o quanto o modelo é surpreso diante de novas informações.

Portanto, agrupar as duplicatas através de tópicos garante um espaço latente organizado pronto para tarefas de classificação, uma vez que essa

representação esclarece as diferenças entre os documentos textuais. No entanto, apesar dessa representação auxiliar no contexto das informações, ainda existem muitas duplicatas que carregam conteúdos de linguagem não natural, principalmente na base Eclipse e NetBeans, resultando em tópicos muito genéricos por compartilharem o mesmo nome de pacotes de *softwares*, nome de classes (código-fonte), tipo de erros, entre outras informações. Um detalhe que acreditamos, além de não inviabilizar o uso dessa representação, pode auxiliar no processamento de contexto para resolver os desafios sobre relatórios de defeitos. Na literatura, Nguyen et al. (2012) adotaram essa estratégia de tópicos para detectar duplicatas, assim como Lukins et al. 2010 no contexto de localização automática de grupo de defeitos em relatórios.

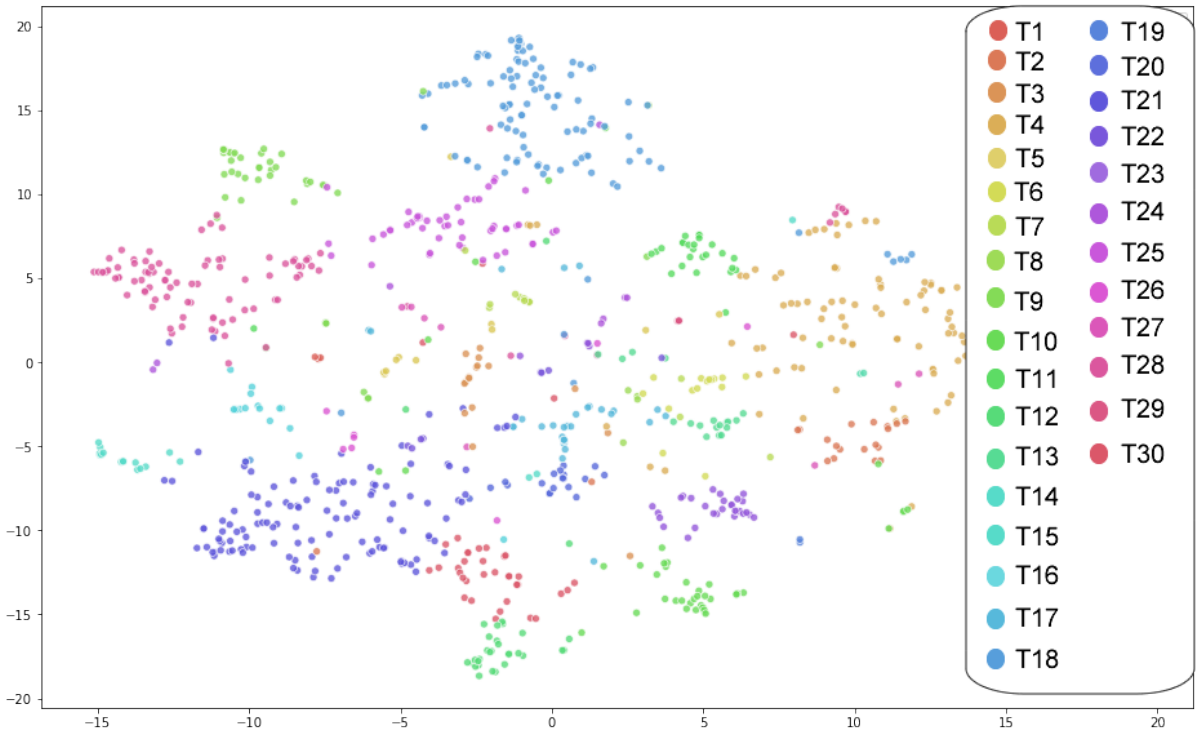


Figura 18 – Visualização latente de 30 tópicos sobre grupos de duplicatas na base do Open Office usando LDA.

Fonte: Próprio Autor

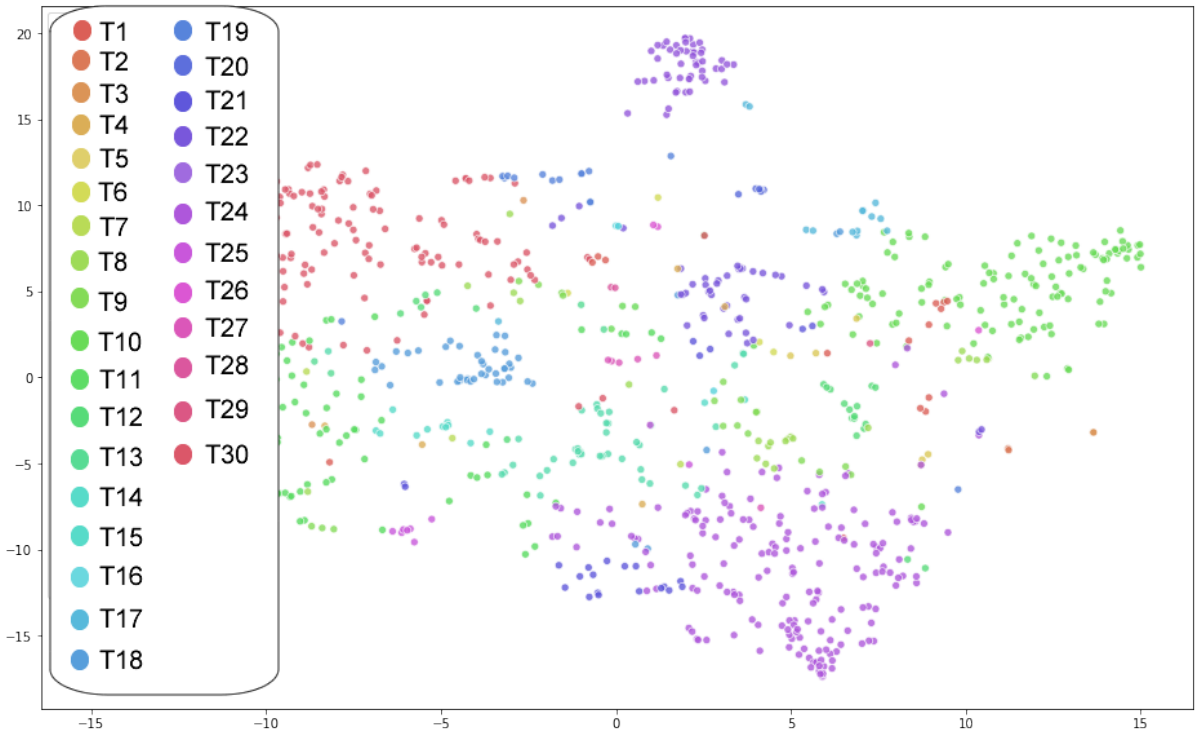


Figura 19 – Visualização latente de 30 tópicos sobre grupos de duplicatas na base do Eclipse usando LDA.

Fonte: Próprio Autor

2.3.3 Embedding: Representação Contínua Distribuída de Palavras

Os *embeddings* são conhecidos como um vetor de características, gerados a partir de uma camada oculta de redes neurais (MANSANO et al., 2018). Eles possuem tamanho fixo e são usados para capturar semântica e sintaxe entre as propriedades das palavras (MIKOLOV et al., 2013a). Essa representação reduz a dimensionalidade de uma informação, descrevendo em um espaço menor as características de uma informação, diferentemente da *tonekinzação* que representa em cada posição da matriz uma palavra. Nesse contexto, Mikolov et al. (2013b) desenvolveram dois modelos semânticos para essa finalidade, o *Skip-gram* e o *CBOW*, ambos ilustrados na Figura 20.

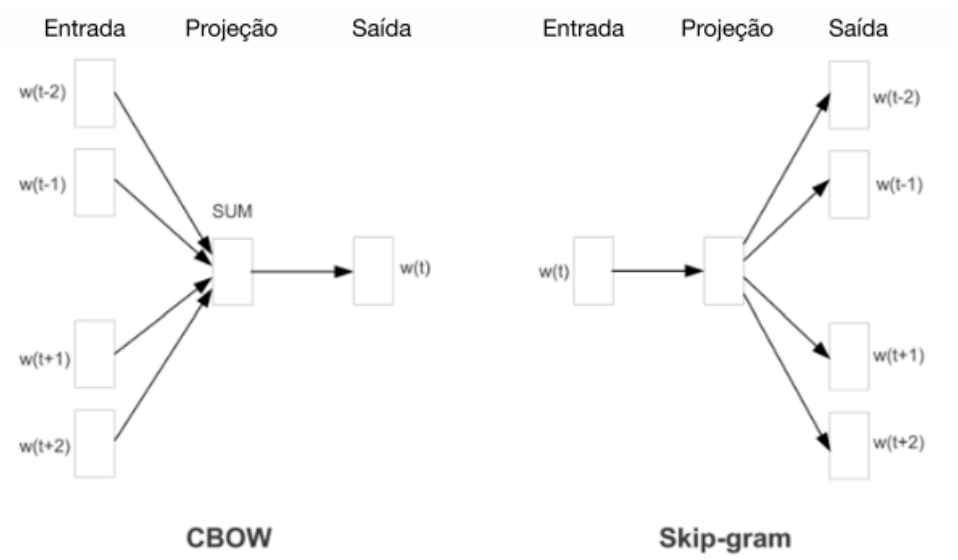


Figura 20 – Modelos CBOW e Skip-gram para representação textual.

Fonte: Mansano et al. (2018)

O método *Skip-gram*, ao receber uma palavra w_i, retorna as palavras $w_i - 2$, $w_i - 1$, $w_i + 1$, $w_i + 2$, ou seja, prevê as palavras que correspondem ao contexto da entrada (MANSANO et al., 2018).

O método *Continuous Bag of Words* (CBOW) funciona de forma inversa, dado uma sequência de palavras $w_i - 2$, $w_i - 1$, $w_i + 1$, $w_i + 2$ o método prevê a palavra w_i, ou seja, prevê a palavra de contexto (MANSANO et al., 2018).

Todas as palavras w são pesos aprendidos por uma rede neural e gerados por uma camada oculta da rede. De posse das representações *embedding* das palavras, podemos visualizar que elas tendem a ficar próximas no espaço. Por exemplo, mesmo aplicando operações algébricas do tipo *Vetor("**Madri**") - Vetor("**Espanha**") + Vetor("**França**")* leva a uma área no espaço latente dimensional próxima do *Vetor("**Paris**")*, demonstrando a relação semântica das palavras de acordo com seu contexto (MANSANO et al., 2018).

No *Skip-gram* e no *CBOW*, há uma lacuna na captura de correlação global entre as palavras. Por exemplo, um texto que trate de viagens deve fornecer palavras como "viagem", "avião" e "turismo", informações correlatas que não serão capturadas, devido à natureza da solução, e por causa da janela deslizante através de n-gramas (MANSANO et al., 2018). Nesse contexto, técnicas como Glove são alternativas para capturar relações globais de coocorrência

das palavras no texto, amenizando o problema de contexto global das palavras (PENNINGTON; SOCHER; MANNING, 2014).

Entre os trabalhos relacionados, Deshmukh et al. (2017) adotaram em sua abordagem o Glove para representar o vocabulário das duplicatas, aprimorando a detecção com esse modelo. No trabalho de Budhiraja et al. (2018a), foram utilizados *embeddings* customizados. Todavia, *embeddigns* baseados no CBOW, Skipgram ou até mesmo o Glove, costumam ser representações estáticas. Pois, são modelos que aprendem sempre com base em uma janela deslizante para combinar a maior de quantidade de palavras dentro de um contexto, isso não possibilita um aprendizado da relação global entre todas as palavras do texto, além de está limitado a um conjunto finito de palavras. Assim, esse sistema de contexto é facilmente colapsado em sentenças com uma mesma palavra que aparece várias vezes no texto com diferentes contextos, como o caso da frase: "*He went to the prison **cell** with his **cell** phone to extract blood **cell** samples from inmates*". No modelo BERT, o vetor para a palavra **cell** será construído de três maneiras diferentes, enquanto que para os outros modelos resultariam em apenas um único vetor. Além disso, a primeira ocorrência da palavra **cell** (*prison cell*), por exemplo, tenderia a está mais próxima de palavras como *incarceration* e *crime*. A segunda palavra **cell** (*cell phone*) estaria mais próxima de palavras como *iphone*, *android* e galaxy (BOMMASANI; DAVIS; CARDIE, 2019).

Portanto, os embeddings gerados pelo modelo BERT demonstram vantagens em relação às representações de palavras ao considerar ordem e contextos dinâmicos. Em comparação a outros modelos, a BERT se adapta mais aos desafios de variações semânticas e contextos em relatórios de defeitos, diante da mistura de vocabulários de linguagem natural e não natural.

2.3.4 Codificação One-hot: Vetorização Binária

Até agora, apresentamos o referencial teórico que envolve o processamento para informações textuais. No entanto, relatórios de defeitos também possuem informações categóricas, como produto, componente, prioridade, severidade, resolução, *status*, e versão. Esses campos precisam ser transformados em

vetores numéricos para conseguirmos compará-los. Todavia, campos como componente ou status, não podem ser alterados puramente para números. Por exemplo, sendo o componente A é igual a 1, e o componente B igual a 2, a transformação numérica pode levar ao aprendizado incorreto de padrões, visto que o componente A pode ser considerado maior que o B. Uma ordem, que passa a ser compreendida pelo modelo, uma vez que transformamos em números. Portanto, uma forma de mitigar o problema é codificar os campos em outro formato vetorial, onde podemos aplicar a codificação *one-hot*, ou em inglês, *one-hot-enconding*, que vetoriza as informações de maneira binária (BROWNLEE, 2018; ZHENG; CASARI, 2018).

Na Tabela 2, ilustramos um exemplo com três valores categóricos para um mesmo atributo de um relatório: componente (*component*) = {*Preferences, General, Downloads Panel*}. Assim, para cada valor do campo componente, cria-se um vetor binário marcando a presença ou ausência da informação de seus valores. Desse modo, identificamos com valor 1 (um) a coluna que o valor categórico está localizado, e as demais colunas devem ser preenchidas com valor 0 (zero).

Tabela 2 – Representação one-hot-encoding

Preferences	**General**	**Downloads Panel**
1	0	0
0	1	0
0	0	1

Com essa representação, os modelos neurais são capazes de correlacionar as categorias, aproximando as duplicatas conforme a similaridade entre os componentes, versões, *status*, severidades e prioridades, além de aprender padrões latentes que não estão claros humanamente na representação *one-hot-encoding*. Além disso, vetores *embeddings* sobre as categorias reduzem a dimensão desses campos, selecionando características relevantes, e codificando em um vetor latente a saída.

2.4 Métricas de Avaliação para Detecção de Duplicatas

A detecção de duplicatas pode ser avaliada de três formas distintas: como recuperação de duplicatas candidatas, como classificação de pares duplicados e como clusterização de duplicatas. Cada avaliação utiliza métricas diferentes que serão apresentadas a seguir.

2.4.1 Recuperação de Relatórios Candidatos

Trabalhos anteriores, como o de Budhiraja et al. (2018a) e do Deshmukh et al. (2017), usam a revocação, ou em inglês *recall*, como métrica para mensurar a qualidade de seus métodos na recuperação de duplicatas, dado um novo relatório como consulta. A saída da consulta é uma lista classificada de possíveis relatórios de defeitos duplicados, ordenados pela probabilidade de serem duplicatas. A taxa de revocação é a proporção de relatórios que possuem pelo menos um relatório duplicado em suas listas de top-k sobre o número total de relatórios avaliados. Para avaliar a lista de top-k relatórios, usamos o Recall@K para mensurar quantas vezes um modelo detecta duplicatas em uma lista de k de relatórios candidatos.

$$Recall@K = \frac{1}{k}\sum_{t=1}^{k}\frac{duplicado}{1} \tag{2.9}$$

$$duplicado = \begin{cases} 1, & \text{Se pelo menos uma duplicata foi encontrada em K relatórios} \\ 0, & \text{Caso contrário} \end{cases}$$

A equação 2.9 expressa a métrica Recall@K, onde o valor K é o tamanho das consultas avaliadas e $duplicado$ é a equação condicional para detectar quando pelo menos um relatório duplicado retornou. Assim mensuramos a frequência com que as duplicatas são encontradas.

2.4.2 Classificação de Pares Duplicados

Em contexto de classificação, o desempenho de um classificador é mensurado por meio das previsões e a classe esperada. Portanto, termos como: *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)*, e *False Negative (FN)*, descre-

vem as previsões do classificador facilitando a visualização do erro (STEHMAN, 1997). Em problemas de duas classes, como a classificação de duplicatas, normalmente é usado a matriz de confusão para visualizar a frequência do erro gerado pelo classificador entre duas classes (SOKOLOVA; LAPALME, 2009; SULEA et al., 2017). Desse modo, depois que treinamos um modelo para classificar um par de relatórios, podemos construir uma matriz de confusão que sumariza o desempenho do classificador, como exemplifica a Tabela 3.

Tabela 3 – Matriz de confusão para classificação de duplicatas.

		Classe prevista	
		Duplicata	Não duplicata
Classe esperada	Duplicata	**7**	3
	Não duplicata	2	**8**

Na Tabela 3, entre 10 relatórios de defeitos, para a classe **duplicata**, o modelo classifica 7 relatórios corretamente (TP), e 3 erroneamente (FN). Para a classe **não duplicata**, o modelo classifica 8 relatórios corretamente (TN), e 2 relatórios erroneamente (FP). Portanto, podemos afirmar com precisão que o modelo na maioria das vezes classifica corretamente, apesar das pequenas margens de erros.

A matriz de confusão é um excelente indicador para mensurar a qualidade do classificador, e com ela é possível calcular outras métricas destinadas à classificação, como a acurácia, precisão, revocação, AOC, dentre outras. A avaliação dos resultados obtidos por este trabalho será feita utilizando essas métricas.

2.4.2.1 Acurácia

Na tarefa de classificação, usamos a medida de precisão tradicional: para cada par de relatórios avaliados, mensuramos a porcentagem de pares classificados corretamente. Deshmukh et al. (2017) avaliaram seus experimentos de

classificação usando essa métrica.

$$Acurácia = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.10}$$

2.4.2.2 Área Sob a Curva Característica de Operação do Receptor (AUROC)

É uma métrica de classificação, usada para distinguir a separabilidade entre duas classes. Ela mensura a relação entre a *True Positive Rate (TPR)* e a *False Positive Rate (FPR)*. O TPR é definido pela equação (2.11). O FPR é definido pela equação (2.12). Por fim, a $Especificidade$ pela equação (2.13).

$$TPR = \frac{TP}{(TP + FN)} \tag{2.11}$$

$$FPR = 1 - Especificidade \tag{2.12}$$

$$Especificidade = \frac{FP}{(TN + FP)} \tag{2.13}$$

O melhor desempenho na AUROC é próximo de uma pontuação de 1, com 0,5 sendo a pontuação de um modelo aleatório que classifica 50% do tempo como positivo e 50% como negativo. Um desempenho ruim nessa métrica acontece quando obtém um valor menor ou igual a 0,5, o que significa que o modelo classifica positivo quando espera classe negativa e negativo quando espera positivo. Usamos a métrica para avaliar a probabilidade de descobrir duplicatas, considerando a sensibilidade sobre os falsos positivos. Budhiraja et al. (2018a) empregaram essa métrica para avaliar qualidade da classificação em seus experimentos.

2.4.3 Proximidade das Duplicatas no Espaço Latente

Agrupar as duplicatas segundo seus mestres fornecem uma noção de grupos, que caracteriza um problema de clusterização. Dessa maneira, é possível mensurar o quão próximos ou bem definidos estão os grupos de duplicatas no espaço latente gerado pelas redes neurais. No entanto, processar todas as características de um relatório é computacionalmente caro. Diante disso, precisamos

de um método para reduzir as dimensões desse documento. Uma alternativa é o método *Distributed Stochastic Neighbor Embedding (t-SNE)*, que visualiza dados de altas dimensões em um espaço bidimensional ou tridimensional (MAATEN; HINTON, 2008). O t-SNE é amplamente usado pela comunidade científica em problemas de clusterização, definido como uma ferramenta de visualização capaz de recuperar grupos bem separados (LINDERMAN; STEINERBERGER, 2019; DEVASSY; GEORGE, 2020).

De uma maneira não supervisionada, visando apoiar a avaliação dos grupos de duplicatas por meio do t-SNE, podemos mensurar a qualidade da densidade dos grupos contra a separabilidade em nossos conjuntos duplicados. Por meio da métrica *Silhouette*, assumimos que existe K grupos nos dados. Logo, na equação de alto nível 2.14, podemos definir o coeficiente *Silhouette*.

$$Silhouette = (b - a)/MAX(a, b) \tag{2.14}$$

Na equação 2.14, a é a distância média entre a amostra do mesmo grupo e b é a distância entre uma amostra e o grupo mais próximo do qual a amostra não faz parte. A *Silhouette* varia de -1 a +1, onde um valor alto indica grupos bem discriminados e um valor negativo pode indicar baixa separabilidade e sobreposições.

2.4.4 Teste-T Student

O teste-t é um dos principais métodos utilizados para se avaliar as diferenças entre as médias de dois grupos, que podem ser independentes ou possuir relação. Trata-se de um teste de hipótese que utiliza conceitos estatísticos para rejeitar ou não uma hipótese nula quando a estatística de teste (t) segue uma distribuição-t de Student (WALPOLE et al., 1993; DAVID; GUNNINK, 1997). Existem três variações do teste t usado para diferentes cenários:

- Amostras independentes: compara as médias de dois grupos.
- Amostra única: testa as médias de um único grupo em relação a uma média conhecida.

- Amostra emparelhada: compara médias do mesmo grupo em diferentes pontos do tempo.

Em todos os casos, o teste produzirá um índice, uma pontuação que é igual à razão entre a diferença média entre dois grupos. Quanto maior o índice, maior será a diferença entre as amostras, o que significa que os resultados do teste são mais reproduzíveis. Todos os resultados dos testes são baseados em determinados valores, chamados de valores-t. Os valores-t são padronizados, calculados a partir de dados amostrais durante um teste de hipótese. Para que possam ser extraídas informações interpretáveis, esses valores precisam ser inseridos em contextos, que consistem em distribuições de probabilidade. No caso das distribuições t, assume-se que as amostras repetidas aleatórias são extraídas de uma população em que a hipótese nula é verdadeira. Para isso, calcula-se o índice de *p-value* como valor de confiança para determinar se é confirmado a hipótese nula entre as distribuições observadas. Geralmente, adota-se valores de 1%, 3% ou 5% como intervalo de confiança. Em relação a este trabalho, os resultados para os experimentos de classificação e recuperação deverão resultar em métricas probabilísticas que poderão ser avaliadas usando o teste-t e confirmar se existe resultados estatisticamente significativos.

2.5 Considerações finais

Neste capítulo foi apresentado um arcabouço teórico para o desenvolvimento do trabalho. A solução proposta no Capítulo 4 se baseia na maioria dos conceitos, além desse ferramental teórico permitir a compreensão dos trabalhos relacionados no Capítulo 3.

O estudo das características de um relatório de defeito foi apresentado na Seção 2.1, permitindo avaliar as principais propriedades do relatório e compreender os desafios de linguagem enfrentados. Na Seção 2.2, apresentamos os diferentes modelos neurais usados para aprendizado de características textuais e categóricas viabilizando o processamento de relatórios. Na Seção 2.3, apresentamos as técnicas de pré-processamento, demonstrando sua importância para modelos neurais. Por fim, na Seção 2.4, descrevemos as principais métricas

empregadas para avaliar a detecção de duplicatas. Nos próximos capítulos são apresentados os trabalhos relacionados e a solução proposta.

3 Trabalhos Relacionados

Neste capítulo são apresentadas as soluções anteriores para detectar relatórios de defeito duplicados, considerando suas abordagens e formas de avaliação, além das características utilizadas. Ao final, fazemos um comparativo entre as abordagens empregadas.

Para a tarefa de detecção de duplicatas podem ser usados recursos textuais ou híbridos, geralmente empregados entre soluções de Recuperação de Informação (RI) e métodos de Aprendizagem de Máquina (AM) / Aprendizagem Profunda (AP). Os recursos de texto são geralmente extraídos de campos como título e descrição para gerar um documento de texto. As soluções híbridas propõem, além do uso de recursos textuais, a utilização de recursos categóricos, como produto, componente, prioridade, severidade do defeito, resolução, status do defeito e versão. Em alguns casos, temos conteúdo não natural como *stack trace*, *logs*, código-fonte, e passos para reproduzir o problema.

3.1 Recuperação de Informação para Detectar Duplicatas

A primeira investigação sobre a detecção de duplicatas foi feita por Runeson et al. (2007), que usaram um vetor baseado em frequência de palavras do documento, construído por meio de métodos como *Bag of Words* (BOW), estes vetores descrevem relatórios de defeitos permitindo aplicar a semelhança de cosseno entre os relatórios para encontrar o candidato duplicado por sua similaridade de conteúdo. Os resultados alcançados obtiveram uma precisão de até 40% na detecção de duplicatas em um conjunto de dados privado da Sony Ericsson Mobile Communications. A principal desvantagem de aplicar BOW é ignorar o contexto de posicionamento da palavra. Ele também considera apenas a frequência das palavras em sua representação vetorial. No entanto, continua sendo um método simples e computacionalmente rápido.

Sun et al. (2010) aprimoram a abordagem anterior usando vetores baseados na Frequência Inversa de Documentos (IDF) para aprender o peso de todas as palavras em um relatório de defeito. As duplicatas candidatas também são

recuperadas por similaridade de cosseno. O corpus nesse trabalho considerou campos como título, descrição e a concatenação de ambos os campos. Em seguida, uma SVM (*Support Vector Machine*) é aplicada a um par de relatórios para classificar se são duplicatas. Seus resultados mostram níveis de precisão entre 50% e 70% em três conjuntos de dados, Eclipse, Firefox e Open Office.

A proposta de Wang et al. (2008) emprega textos naturais e não naturais extraídos de campos como descrição, *stack traces* e *logs*. A ideia é calcular duas pontuações de similaridades combinando informações naturais e não naturais de relatórios de defeitos, vetorizando por meio da Frequência do Termo na Frequência Inversa de Documentos (tf-IDF) e recuperando candidatos por similaridade de cosseno. Eles atingiram uma precisão de 71% e 82% para detectar as duplicatas nos conjuntos de dados Eclipse e Firefox. Utilizar evidências de linguagem não natural, apesar de aprimorar os resultados na detecção das duplicatas, são informações não disponíveis em bases públicas como a base de Lazar et al. (2014). A utlização de técnicas para processamento de texto não natural podem permitir separar esse conteúdo e assim avaliar os relatórios segundo suas semelhanças com base nessa informação.

Jalbert e Weimer (2008) desenvolveram um classificador combinando severidade do defeito, campos de data, pontuação de similaridade com campos de texto e grafos com links entre os defeitos para discriminar duplicatas. Sua estratégia alcançou 43% de precisão no conjunto de dados, Mozilla.

Sun et al. (2011) propuseram o REP, um método de recuperação baseado no BM25F$_{ext}$ para vetorizar campos textuais e categóricos. O método suporta consultas de texto longo e aprende pesos através do gradiente descendente estocástico. O BM25F foi desenvolvido para solucionar a falta de normalização e padronização em todos os campos do BM25. A proposta alcançou 68% de 72% de precisão nos conjuntos de dados, Eclipse, Open Office e Mozilla.

Nguyen et al. (2012) usaram o BM25F$_{ext}$ combinado com LDA, um modelo de tópico que descreve as duplicatas com base em tópicos compartilhados, calculando a pontuação de similaridade entre os defeitos. O vetor de recurso gerado pelo LDA é a distribuição de probabilidade de tópicos em um relatório de defeito, uma estratégia que alcançou em torno de de 80% nos conjuntos de dados, Eclipse, Open Office e Mozilla.

Aggarwal et al. (2015) criaram um método contextual usando um vocabulário específico extraído dos livros de engenharia de software e documentos técnicos. Essa abordagem produz um corpus mais rico, usando o BM25F para calcular a pontuação de similaridade entre relatórios de defeitos, detectando 90% de duplicatas em Android, Open Office, Mozilla e Eclipse.

Em contraste com as abordagens de RI, a maioria das soluções tende a representar relatórios de defeitos usando a frequência de palavras ou distribuições de tópicos, extraindo sintaxe e contexto do vocabulário, identificando assim as duplicatas similares. Apesar de o desempenho desses métodos serem rápidos, a ordem das palavras é descartada na representação. Desse modo, as palavras não possuem o significado semântico. Uma alternativa para essa estratégia é o uso do LDA com a representação em tópicos. No entanto, essa representação fornece um contexto latente sobre o vocabulário do documento, e não individualmente para cada palavra do texto.

Pesquisas como a de Jalbert e Weimer (2008) incentivaram o uso dos atributos categóricos em conjunto com o text, para impulsionar a detecção. Entretanto, nenhuma solução baseada em RI resolveu o problema do contexto das palavras no vocabulário de relatórios. Além disso, o número de duplicatas experimentado foi pequeno, normalmente selecionado por período ou número máximo de duplicatas, devido a complexidade do processamento de muitas duplicatas. Se a seleção não for representativa, o subconjunto selecionado pode mascarar o resultado (DASZYKOWSKI; WALCZAK; MASSART, 2002). A pesquisa de Lazar et al. (2014), fornecem pares de duplicatas precisamente analisados e preparados para classificação de duplicatas através de heurísticas e relações em grafos, fornecendo um conjunto de duplicatas suficiente para métodos que precisem processar uma quantidade razoável de duplicatas.

Na próxima seção, apresentamos outra linha de pesquisa, que envolve soluções que detectam duplicatas baseadas em aprendizagem de máquina. As abordagens aprendem o contexto semântico sobre relatórios de defeitos automaticamente, construindo representações latentes sobre esses documentos.

3.2 Aprendizagem Profunda para Detectar Duplicatas

Budhiraja et al. (2018a) propuseram um modelo de classificação profunda baseado em pares de relatórios para detectar se um par é duplicado ou não, nomeado sua estratégia como *Deep Word Embedding Neural Network* (DWEN). O modelo usa o título e a descrição dos relatórios de defeitos. As palavras do documento são representadas por vetores Skip-gram. A classificação é feita usando uma camada MLP e uma camada de saída Sigmoid para discriminar se o par é duplicado ou não. Eles relataram recuperar 80% das duplicatas no conjunto de dados Open Office e Firefox. Nessa abordagem não há uma avaliação para tarefas de classificação. Além disso, o trabalho não usa recursos textuais baseados em tópicos, além do aprendizado não empregar nenhum mecanismo de atenção. Essa abordagem, apesar de eficiente, pode ser aprimorada se os recursos categóricos fossem adicionados a detecção das duplicatas em conjunto com o texto, permitindo que a MLP selecione as melhores características sobre um relatório para representá-las em um novo espaço latente.

Deshmukh et al. (2017) usam uma arquitetura Siamesa profunda, recebendo uma entrada híbrida, com recursos textuais (título, descrição) e categóricos (versão, componente, produto). A rede Siamesa é composta por três modelos: MLP, BiLSTM e CNN. A saída do processamento desses modelos é combinada em um vetor *embedding* que representa o relatório. A rede é treinada usando a função de perda baseada no terceto, a *Triplet Loss*, que em cada instância usa um candidato, um exemplo negativo e um positivo, com o objetivo de maximizar a semelhança entre duplicatas e minimizar entre não duplicatas. Os autores atingiram uma precisão entre 79% e 81% nos conjuntos de dados Eclipse, NetBeans e Open Office. Comparada com o nosso trabalho, a proposta de Deshmukh et al. (2017) é a abordagem mais semelhante à utilizada em nosso trabalho, pois os autores usaram a arquitetura Siamesa, recursos textuais e categóricos, além de avaliarem tanto em tarefas de classificação quanto de recuperação. No entanto, empregam redes diferentes em sua arquitetura, redes que não consideram o uso de mecanismos de atenção na aprendizagem. Além disso, os autores não utilizaram representação baseada em modelagem

de tópicos.

Em referência aos trabalhos de AM, as soluções instauraram uma nova maneira de produzir representações para relatórios de defeitos, construindo características latentes em um espaço que reúne significado semântico sobre os relatórios. Desse modo, as diferentes redes neurais são capazes de receber campos textuais e categóricos, gerando *embedding* que servem como um vetor de descrição dos relatórios, permitindo compará-los e descobrir os documentos similares. Entretanto, as redes adotadas pelos trabalhos anteriores, não empregam mecanismos de atenção ou aprendizagem de contexto dinâmico para palavras. Diante disso, detectar as mudanças de contexto dentro de textos em relatórios, acreditamos enriquecer a representação latente dos documentos na presença de conteúdos não naturais na descrição.

3.3 Síntese dos Trabalhos Relacionados

A Tabela 4 resume as principais características dos trabalhos relacionados neste capítulo, expondo os modelos, informações utilizadas para detecção e os resultados. Na última coluna, são apresentadas as bases de relatórios que foram utilizadas pelos trabalhos, além das métricas e resultados.

Entre os trabalhos relacionados o que obteve maior resultado para recuperar duplicatas foram Aggarwal et al. (2015), e poucos trabalhos realizam a comparação direta entre eles devido à alta variedade de bases distintas para avaliar a tarefa de detecção de duplicatas. Entre as bases, as mais comuns segundo Lazar et al. (2014), são Eclipse, NetBeans e Open Office que compreendem um histórico de mais de 10 anos de relatórios coletados.

Os trabalhos de Budhiraja et al. (2018a) e Deshmukh et al. (2017) são os *baselines* dessa pesquisa marcados com um asterisco na Tabela 4, justamente, por serem abordagens mais semelhantes à proposta desta pesquisa. Nesse sentido, Budhiraja et al. (2018a) realizaram experimentos para comparar diretamente os resultados com trabalhos relacionados baseados em RI, como o Aggarwal et al. (2015), demonstrando o ganho e o impacto de soluções baseadas em AP para construção de representações que impactam positivamente em tarefas de detecção das duplicatas. Portanto, seguindo essa mesma linha de

pesquisa, adotamos somente os trabalhos baseados em AP como *baselines*, e no Capítulo 5 apresentamos os resultados da nossa fase de experimentação.

Tabela 4 – Comparação dos trabalhos relacionados (Recall@K com K entre 5 e 25)

Autor	**Informação**	**Modelo**	**Recall@K**
Runeson et al. (2007)	Textual	TF/IDF	31-42% (Privado)
Wang et al. (2008)	Híbrida	TF/IDF	65-82% (Eclipse) 42-71% (Firefox)
Jalbert e Weimer (2008)	Híbrida	TF/IDF Grafo	27-43% (Mozilla)
Sun et al. (2010)	Textual	TF/IDF SVM	31-58% (Open Office) 39-69% (Firefox) 35-59% (Eclipse)
Sun et al. (2011)	Híbrida	BM25F$_{ext}$	38-68% (Open Office) 38-68% (Mozilla) 42-72% (Eclipse) 38-69% (Large Eclipse)
Nguyen et al. (2012)	Híbrida	BM25F LDA	40-81% (Open Office) 40-80% (Mozilla) 59-82% (Eclipse)
Aggarwal et al. (2015)	Híbrida	BM25F Contexto	87-92% (Android) 84-91% (Open Office) 85-92% (Mozilla) 85-92% (Eclipse)
* Deshmukh et al. (2017)	Híbrida	MLP Bi-LSTM CNN	50-79% (Open Office) 55-81% (Net Beans) 61-81% (Eclipse)
* Budhiraja et al. (2018a)	Textual	MLP	21-77% (Open Office) 25-70% (Firefox)

* Trabalhos selecionados como *baseline* desta pesquisa.

3.4 Considerações finais

Antes do advento das soluções baseadas em AM, era necessária uma etapa de extração das características nos relatórios, que servia para criar representações do relatório permitindo compará-los e calcular a similaridade entre eles. Atualmente, os trabalhos que envolvem AM realizam a etapa de extração automaticamente, o que tornou as soluções mais focadas em modelos e menos na engenharia de atributos. Por fim, o objetivo de todas as soluções são encontrar uma forma de representar os relatórios de defeitos, para compará-los e classificá-los como duplicado ou não. Portanto, as soluções baseadas em AM se

apresentam páreo com o estado da arte em RI. Isso mostra a tendência desses novos modelos de aprendizagem e o espaço para mudanças que superem o estado da arte. Em contrapartida, nenhuma solução avaliou o uso de abordagens baseadas em RI e AM ao mesmo tempo, combinando diferentes representações sobre os atributos para que possam ser aprendidas por modelos neurais para selecionar as melhores características, além do uso de mecanismos de atenção.

4 Uma Abordagem para Detectar Relatórios de Defeitos Duplicados

Neste capítulo descrevemos uma estrutura para a detecção de relatórios de defeitos duplicados, bem como um novo modelo de representação de relatórios baseado em aprendizagem profunda, o SiameseQAT. Normalmente na literatura é usual formular a tarefa de detecção de relatórios duplicados como ou uma tarefa de recuperação de duplicatas ou de classificação binária. No entanto, em nosso trabalho mostraremos que nossa estrutura pode ser aplicada para ambas as tarefas. Além disso, descrevemos desde a obtenção dos relatórios de defeitos até a tarefa de detecção das duplicatas.

O núcleo de nossa abordagem é usar o SiameseQAT para gerar representações de relatórios de defeitos e usá-los em tarefas de recuperação e classificação. Na fase de treinamento do modelo, cada instância de entrada é um quinteto contendo um relatório aleatório (a âncora), um duplicado (o exemplo positivo), um relatório não duplicado (um exemplo negativo), além de dois elementos extras: uma representação do grupo de todas as duplicatas do exemplo positivo (o centroide positivo) e o mesmo para o grupo de todas as duplicatas do exemplo negativo (o centroide negativo).

O quinteto é usado para treinar o SiameseQAT para aprender representações de relatórios de defeitos, maximizando e minimizando a semelhança entre representações duplicatas e não duplicatas, conduzidas pelas funções *Quintet Loss* se estiver treinando uma tarefa de recuperação, e entropia cruzada binária para classificação.

Na Figura 21, detectamos as duplicatas através de três fases principais: (1) *treinamento do modelo*, (2) *recuperação das duplicatas* e (3) *classificação das duplicatas*.

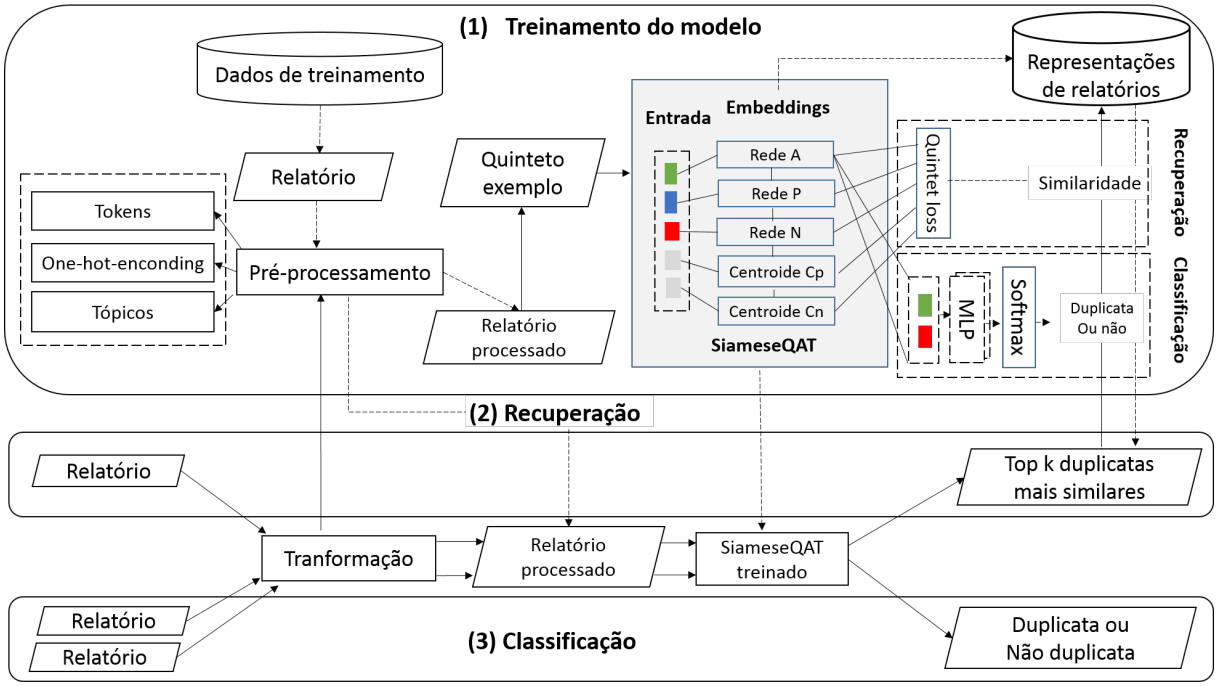


Figura 21 – Visão geral da arquitetura para detectar relatórios duplicados, dividida em três fases: (1) treinamento do modelo; (2) recuperação; e (3) classificação.

- **Fase 1 - Treinamento do modelo**: nesta fase, treinamos o SiameseQAT usando exemplos de relatórios de defeitos. Primeiro, eles são pré-processados, extraindo tokens, tópicos e representações de codificação *one-hot* para informações categóricas. Após esse pré-processamento, esse conjunto de dados pode ser usado para gerar exemplos de quintetos para o treinamento de nossos modelos (os detalhes dos modelos são melhor explicados na seção 4.4). O modelo de recuperação é treinado primeiro para gerar os *embeddings* dos relatórios de defeitos. Em seguida, o modelo de classificação é treinado usando os mesmos *embeddings* gerados pelo modelo de recuperação, ou seja, os pesos do modelo SiameseQAT treinado para recuperação são usados para gerar a entrada do modelo classificador binário. O resultado desse estágio são dois modelos de redes neurais, um para gerar *embeddings* de relatórios de defeitos e outro para classificar dois relatórios. Além desses modelos, as representações *embeddings* de todos os relatórios são usadas no treinamento também armazenadas em um conjunto de dados para futuras tarefas de recuperação. Nas próximas seções descrevemos com mais detalhes sobre a construção do quinteto,

as etapas do treinamento do modelo, pré-processamento, além da importância dos embeddings gerados pelo SiameseQAT, destacando a função de perda proposta nesta pesquisa, a Quintet Loss.

- **Fase 2 - Recuperação das duplicatas**: nesta fase, a estrutura recebe como entrada um novo relatório de defeito e precisa recuperar os relatórios top-k com maior probabilidade de serem uma duplicata da entrada. Para fazer isso, o novo relatório deve primeiro gerar sua representação *embedding*, passando pelo mesmo processo de pré-processamento e usando SiameseQAT. Essa representação *embedding* é usada para recuperar os relatórios de defeitos duplicados prováveis de serem duplicatas para a consulta. Com isso, um *triager* (ou um usuário, ou um método de classificação automática) pode identificar se esses candidatos são de fato réplicas.

- **Fase 3 - Classificação das duplicatas**: nossa estrutura também pode ser usada para determinar automaticamente se um par de relatórios é duplicado ou não. Para fazer isso, os relatórios também devem ser pré-processados e ter seus *embeddings* gerados pelo modelo SiameseQAT. Em seguida, o classificador binário treinado anteriormente pode ser usado para avaliar se eles são de fato duplicados. Ao usar o mesmo modelo para gerar a representação *embedding* para as tarefas de recuperação e classificação, os *embeddings* de relatórios armazenados anteriormente podem ser usados sem repetir todo o processo, simplificando a comparação de um número maior de relatórios.

4.1 Dados de Treinamento

À aquisição dos dados de treinamento é feita por meio do trabalho de Lazar et al. (2014), que fornecem três grandes conjuntos de projetos de software aberto, populares para tarefa de detecção das duplicatas. As bases Eclipse, Open Office e Netbeans, contemplam registros de defeitos reportados por milhões de usuários, com aproximadamente **500 mil registros** de defeitos, além de englobar relatórios sobre ambientes de desenvolvimento em Java e sobre a

ferramenta Open Office. As bases viabilizam a execução de experimentos e as próximas etapas de treinamento do modelo e detecção das duplicatas.

É importante salientar que os conjuntos de dados de Lazar et al. (2014) contêm informações sobre relatórios de defeitos, o gabarito das duplicatas e uma informação que aponta o ID do relatório de defeito duplicado. Com isso, podemos treinar modelos supervisionados, principalmente para classificar duplicatas.

Em bases de relatórios duplicados, a divisão em treino e teste precisa ser feita baseada na combinação entre os pares de relatórios duplicados, assim como é feito a divisão no problema de pares de perguntas duplicatas (SHARMA et al., 2019). Nesse sentido, existem duas abordagens para divisão dos pares: combinação de pares únicos ou a construção de pares únicos com o mestre. Por exemplo, em um grupo de 4 duplicatas, usando a primeira estratégia, podemos construir 6 pares únicos através da combinação entre as duplicatas desse mesmo grupo. Na segunda estratégia, podemos construir 3 pares únicos combinando somente o relatório mestre contra todas as duplicatas desse mesmo grupo. Na Figura 22, demonstramos às duas abordagens e a proporção de pares em cada abordagem.

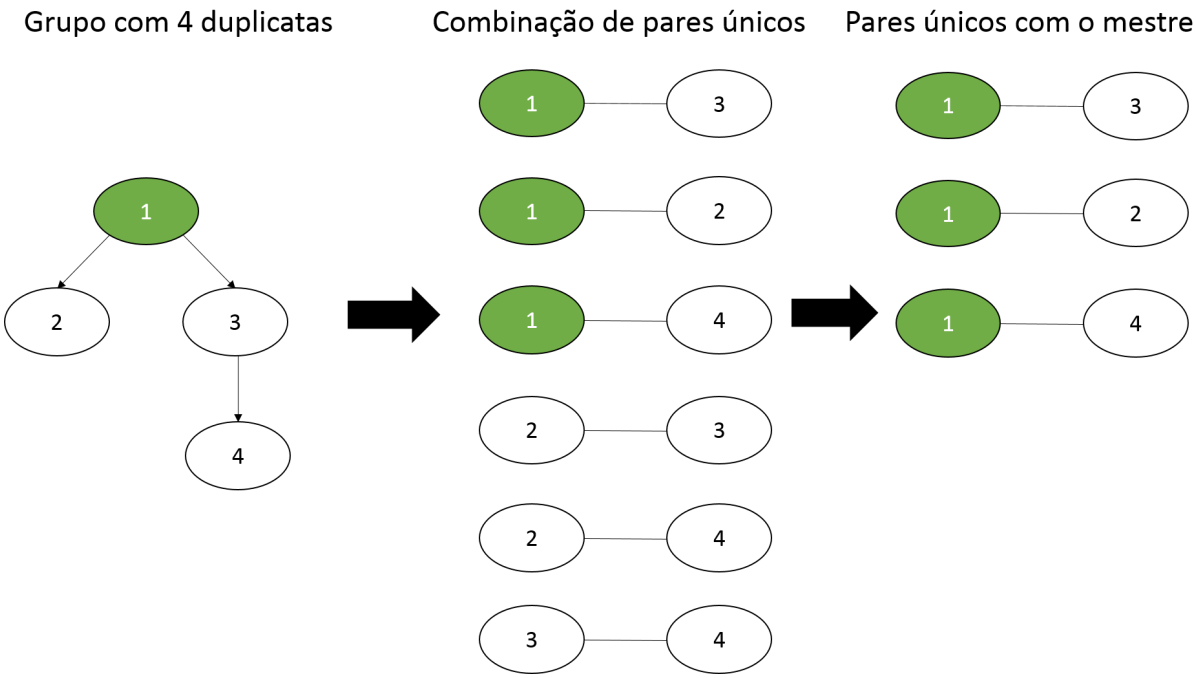


Figura 22 – Divisão das duplicatas em pares.

Como observado na Figura 22, a construção de pares únicos com o mestre representa a estratégia mais barata computacionalmente, além de funcionar de forma aproximada em relação a outra estratégia de combinar todos os pares possíveis, que possui uma representatividade maior nos dados, mas aumenta consideravelmente a quantidade de informação para o treinamento. Dessa maneira, dependendo do tamanho do grupo de duplicatas, a estratégia de combinar todos os pares pode ser inviável computacionalmente. Portanto, combinar com o mestre é a nossa estratégia utilizada para gerar e construir a divisão dos pares. Nessa divisão, separa-se **90%** dos pares para **treino** e **10%** para **teste**, conforme pode ser observado na Tabela 5.

Separado os pares de relatórios, podemos aplicar dois tipos avaliações usando as duplicatas no teste. A divisão cega e a divisão disjunta. Segundo Sharma et al. (2019), às duas avaliações simulam cenários do mundo real que melhor caracterizam os pares duplicados. Portanto, cada avaliação funciona da seguinte maneira:

- **Divisão cega**: divide o conjunto de dados enquanto preserva o balanceamento (a proporção de duplicatas e não duplicadas) em cada novo conjunto de dados. Para essa abordagem, ignora-se o fato de que as duplicatas no conjunto de treinamento também podem aparecer nos conjuntos de teste.
- **Divisão disjunta**: essa divisão preserva o balanceamento dos dados e garante que as duplicatas que aparecem nos conjuntos de treinamento e teste sejam distintas.

Para treinar nosso modelo proposto, primeiro treinamos usando a divisão disjunta na fase de recuperação das duplicatas. Depois, treinamos usando a divisão cega na fase de classificação das duplicatas. Conforme Sharma et al. (2019) afirmam, a divisão disjunta torna a tarefa mais difícil e pode simular um problema de transferência de aprendizado, pois o vocabulário dos conjuntos de teste são diferentes do vocabulário do conjunto de treinamento, forçando o modelo a generalizar mais suas representações aprendidas. No caso da divisão cega, os relatórios duplicados tendem a compartilhar bastante vocabulário,

portanto acreditamos que essa divisão representa melhor uma avaliação dos relatórios em um cenário do mundo real.

Tabela 5 – Divisão em treino (90%) e teste (10%) para as bases.

Base	**Pares de Treino**	**Pares de Teste**	**Total de Relatórios**
Eclipse	773.181	28.025	334.422
Net Beans	1.280.106	35.025	216.715
Open Office	1.226.059	23.065	72.234
Total	**3.279.346**	**86.115**	**623.371**

Além da divisão aplicada nos relatórios, as duplicatas são agrupadas por meio do atributo **dup_id** nas bases. Nesse agrupamento, um dos relatórios duplicados é eleito como "mestre", uma escolha feita por meio do mais antigo ou escolha por um especialista (SUN et al., 2011). Para verificar essa informação, todos os relatórios que não possuem nenhum valor no campo **dup_id** representam o relatório mestre. A Figura 23 ilustra um exemplo da organização das duplicatas facilitando a construção do conjunto de treino e teste.

DUPLICADOS
MESTRE 1: DUP-1.1, DUP-2.1,..
MESTRE 2: DUP-2.1, DUP-2.2,..
MESTRE 3: DUP-3.1, DUP-3.2,..
...
MESTRE M: DUP-M.1, DUP-M.2,..

Figura 23 – Organização das duplicatas em grupos.

Para processar os relatórios é preciso de algum mecanismo que possibilite a leitura do conjunto de relatórios por meio de consultas. Todos os relatórios em formato tabular são organizados por linhas e colunas, para facilitar o processamento dos documentos. As bases possuem uma coluna de nome **dup_id**, que referencia o **id** do relatório duplicado. Além disso, possuem atributos que descrevem componentes de *software* como (**component**), versão (**version**), além do título (**short_desc**) e descrição (**description**) que descrevem o problema. Caso o campo **dup_id** esteja vazio, isso significa que o relatório não é duplicado de ninguém e consiste em um mestre. A Tabela 6 apresenta três exemplos de relatórios em formato tabular.

Tabela 6 – Três exemplos de relatórios em formato tabular.

id	**short_desc**	**long_desc**	**component**	**version**	**dup_id**
10954	Dialup properties needs to be exposed in prefs..	The dialup properties of the profile should be...	Preferences	Thunk	[ ]
14871	[Find] Find whole word only...	Please add Match Whole Word Only option to bro...	General	Thunk	269442
955893	browser download useToolkitUI=true doesnt work...	User Agent: Mozilla/5.0 (Windows NT 6.1; rv:24...	Downloads Panel	24 Branch	926146

Tabela 7 – Informações adicionais sobre a base de dados fornecida por Lazar et al. (2014).

Coluna	**Informação**	**Descrição**
bug_id	numérica	ID do relatório
product	categórica	Nome do produto de software
description	textual	Descrição longa sobre o problema contendo passos para reproduzir o problema, *stack traces* e código-fonte juntamente com texto natural
bug_severity	categórica	Severidade do problema definido em categorias
dup_id	numérica	ID do relatório duplicado
short_desc	textual	Um texto curto escrito de forma livre para caracterizar o problema
priority	categórica	Prioridade definida para o problema
version	categórica	Versão do software, modulo ou componente
component	categórica	Nome do componente de *software*
delta_ts	data	Data de resolução do problema
bug_status	categórica	Status do problema
creation_ts	data	Data de criação do problema
resolution	categórica	*Status* da resolução do problema

A lista completa dos atributos disponibilizados por Lazar et al. (2014) é descrita na Tabela 7. Em nossa abordagem, não utilizamos todos os atributos, tais como: **creation_ts**, **delta_ts**, **dup_id** e **bug_id**. Os campos citados foram descartados devido à alta variância presente em seus valores.

Nas próximas seções, descrevemos em detalhes como cada atributo do relatório será processado.

4.2 Pré-Processamento

Durante esta fase preparamos os conjuntos de relatórios como entrada para o SiameseQAT. As informações textuais e categóricas de um relatório precisam passar por transformações de padronização que resultem em vetores numéricos e controlados, tamanho fixo (GOODFELLOW et al., 2016). Portanto, para os atributos de textos, aplicaremos tokenização e extração de tópicos. Para os atributos categóricos, aplicaremos a codificação *one-hot*.

4.2.1 Processamento Textual

Para atributos textuais como título (**short_desc**) e descrição (**description**), aplicamos uma tokenização, transformando as palavras em tokens numéricos. Dessa maneira, assumindo a frase "*Out of memory error with Mylar?*", extraída da descrição de um relatório. Primeiro, transformamos todas as palavras para caixa-baixa. Em seguida, aplicamos a tokenização BERT, para separar por espaço cada palavra da sentença, construindo pseudopalavras OOV que não pertencem ao domínio da BERT. Então, adicionamos os tokens [CLS] e [SEP], no início e fim da sentença. Por fim, mapeamos cada palavra em token único dentro de um vetor numérico. O fim a fim do processamento destacamos nos vetores de exemplo em 4.1, 4.2, 4.3.

$$[out, of, memory, error, with, mylar?] \tag{4.1}$$

$$[out, of, memory, error, with, my, \#\#lar, ?] \tag{4.2}$$

$$[101, 9683, 232, 11, 166, 812, 298, 301, 33, 102] \tag{4.3}$$

O texto de um relatório (***short_desc*** e ***description***) possui tamanho variável (como apresentado na Tabela 9), ou seja, precisa passar por uma padronização, visto que as redes neurais só trabalham com vetores de tamanho fixo e numéricos. Logo, como aplicar somente a tokenização não é suficiente, é necessário o uso do *padding* como técnica complementar. O *padding* força a construção de sequências com tamanho normalizado, estabelecendo limites no vetor de entrada, que podem preencher com zeros as posições vazias à frente, ou aquelas após o vetor. Portanto, aplicando *padding* de 10 no mesmo exemplo 4.1, temos o seguinte vetor de saída 4.4, que força o tamanho da sequência a ter sempre 10 posições.

$$[101, 9683, 232, 11, 166, 812, 298, 301, 33, 102, 0] \tag{4.4}$$

Analisando a frequência de palavras (*tokens*) destacado na Tabela 9, pode ser notado que *short_desc* não ultrapassa um total de 20 palavras em média, enquanto *description*, pode conter entre 200 ou 700 palavras na maioria dos relatórios, justificando uma normalização como *padding*. Para o texto dos relatórios, consideramos as 100 primeiras palavras para título e descrição, após considerar 20, 50, 100 e 200 total de palavras.

Além da tokenização e o *padding*, treinar uma rede neural do zero não é considerado uma boa prática, e normalmente na literatura é usado *embeddings* pré-treinados, que resolvam a partida a frio de modelos de aprendizagem e tenham um vocabulário grande de palavras. Essa técnica, enriquece o conhecimento do modelo em linguagem de texto, além de incorporar um conhecimento pré-computado no modelo. Para relatórios de defeitos, a técnica aprimora o conhecimento da rede sobre termos em inglês, já que os relatórios são escritos nesse idioma. Na literatura, os modelos pré-treinados mais comum são a BERT (JAIN; WALLACE, 2019), o *Fast Text* (JOULIN et al., 2016), e o Glove (PENNINGTON; SOCHER; MANNING, 2014). Como já destacamos na seção 2.3.3, os *embeddings* da BERT são os mais adaptados para contextos dinâmicos sobre as palavras, e dois modelos são avaliados como alternativas, o

BERT_{BASE} e BERT_{LARGE}. Por questões de recursos, adotaremos o BERT_{BASE} como *embeddings* pré-treinados.

Também aplicamos a representação baseada em modelagem de tópicos sobre o texto dos relatórios, combinando o título e a descrição dos relatórios em um campo único, de forma que, possamos extrair os tópicos usando o método LDA para esse campo único. No fim, a distribuição em tópicos para cada relatório foi experimentada com tamanhos de 10, 20, 30 e 50, e obtivemos o melhor resultado com tamanho 30. Essas configurações são exaustivamente testadas através do Grid Search[1], resultando na melhor configuração.

4.2.2 Processamento Categórico

Para os atributos categóricos de relatórios de defeitos, como produto, componente, prioridade, severidade, resolução, *status*, e versão, aplicamos a codificação *one-hot*, transformando cada campo em uma matriz esparsa com valores binários, entre 0 e 1, que identificam quando há a presença de informação nos atributos. Na Tabela 8, apresentamos um exemplo para os relatórios de ID=$\{10954, 14871, 955893\}$. O símbolo **XXXX** nas colunas, identifica qualquer valor para a coluna categórica. O *component*, por exemplo, seguindo o padrão *component_*{**XXXX**}, temos valores: *Preferences, General, e Download Panel*. Assim, os valores são transformados em colunas, para: *component_*$Preferences$, *component_*$General$ e *component_*$Download\_Panel$. A mesma lógica é aplicada para todos os atributos categóricos mencionados anteriormente.

Tabela 8 – Processamento *one-hot-encoding* para os atributos categóricos.

bug_id	**version_Thunk**	**version_25_Branch**	**component_XXXX**
10954	1	0	1
14871	1	0	0
955893	0	1	0

[1] https //scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

4.3 Treinamento do Modelo

Nesta etapa, o SiameseQAT é treinado usando quintetos e duas funções de perda. Primeiro através da *Quintet Loss*, em seguida, por meio da entropia cruzada binária. Dessa forma, treinamos o modelo para classificar pares de duplicatas e comparar instâncias duplicatas contra exemplos não duplicados, descobrindo a semelhança entre os relatórios, e automaticamente aprendendo a representar relatórios de defeitos. A tarefa de *recuperação das duplicatas* é feita por meio da função *Quintet Loss*, que será definida nas próximas seções, enquanto a tarefa de *classificação das duplicatas*, é orientada através da entropia cruzada binária. Como resultado desse estágio, os *embeddings* do modelo treinado são gerados para todos os relatórios no conjunto de dados, produzindo a base de representações dos relatórios.

O treinamento do SiameseQAT é feito em blocos (ou *batches*), por causa do recurso computacional requerido se for feito sobre todos os relatórios da base (GOODFELLOW et al., 2016). Nesse sentido, construímos blocos de relatórios para treinar o modelo, construindo n triplas $T = \{T_1, T_2, T_3, ..., T_N\}$, que são definidas como um conjunto de relatórios relacionados, formado por um aleatório A (âncora), um exemplo positivo P (duplicata) de A, e outro negativo N (não duplicata) de A. Com isso, adicionamos dois *centroides* para cada tripla do bloco. O primeiro chamado de C_P, para as duplicatas positivas de A. O segundo denominado C_N, para as duplicatas negativas de A. Assim, dizemos que um *centroide* é definido como a média dos vetores em todos os relatórios em um determinado grupo de duplicatas. Em seguida, um quinteto de relatórios, pode ser definido como um grupo Q, de cinco relatórios relacionados, composto pela combinação de uma instância A, P, N, além dos *centroides* C_P e C_N, definido como Q = $\{A, P, N, C_P, C_N\}$. O quinteto é a entrada para o SiameseQAT usando a função de perda *Quintet Loss* durante o primeiro treinamento para recuperar duplicatas. Na Figura 24, destacamos a diferença do bloco de treinamento construído pela função *Triplet Loss* em comparação a *Quintet Loss*.

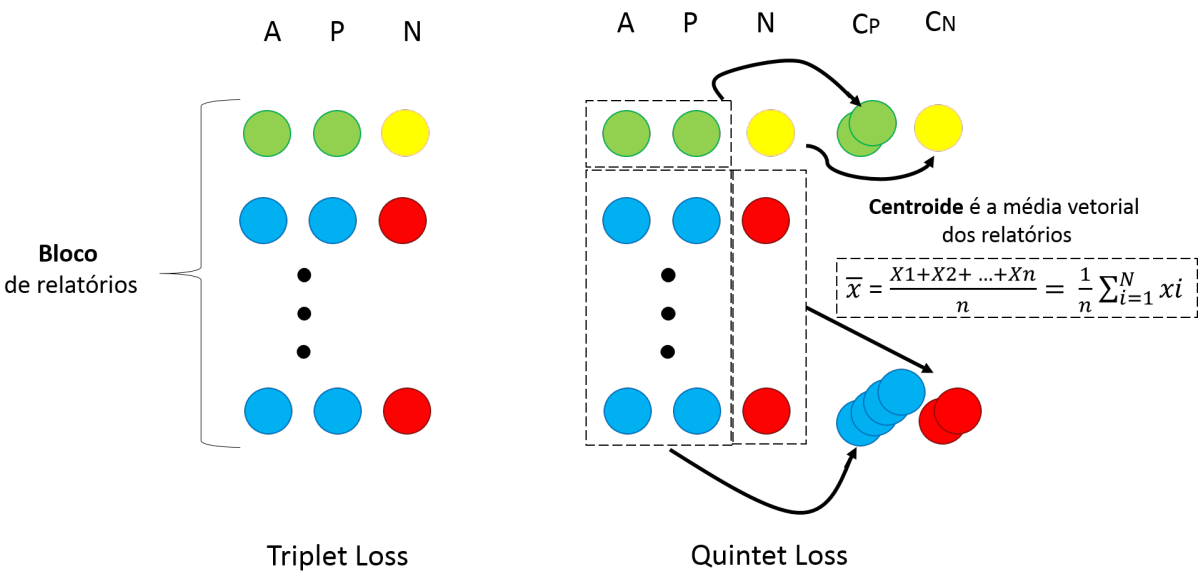


Figura 24 – Construção do bloco de treinamento na *Triplet Loss* e *Quintet Loss*.

Na Figura 24, os centroides podem ser gerados a partir da combinação de uma única ou muitas duplicatas do mesmo grupo, por meio da combinação de exemplos entre os tercetos. Nesse cenário, não há garantias que o centroide encontrado representa a informação global de todas as duplicatas deste grupo, pois pode existir grupos com n duplicatas, onde o bloco de treinamento gerado, possui um número inferior ou igual a população total de duplicatas que pertecem a um determinado grupo. Portanto, o centroide treinado na *Quintet Loss*, otimiza o aprendizado do modelo SiameseQAT a convergir as duplicatas de acordo com o centroide local encontrado nesse bloco de treinamento, além de tentar aproximar as duplicatas e o centroide local do seu centroide global que representa todas as duplicatas de um mesmo grupo.

Para tarefa de *classificação das duplicatas*, um segundo treinamento é efetuado repassando como entrada para o SiameseQAT, pares de relatórios, que podem ser duplicado ou não. Portanto, o bloco de treinamento pode ser constituído por instâncias $\{A, P\}$, ou instâncias $\{A, N\}$. Dessa forma, o modelo aprende a classificar os pares usando a função de entropia cruzada binária e a *Softmax* para gerar em duas probabilidades a porcentagem de ser duplicata.

Na próxima seção, descrevemos o modelo SiameseQAT, destacando o aprendizado de características sobre relatórios de defeitos, por meio da entrada de quintetos.

4.4 SiameseQAT: Detecção Duplicata Utilizando Aprendizagem Baseada em Contexto Semântico e Informações Coletivas Duplicatas em uma Abordagem Siamesa

SiameseQAT é uma rede neural siamesa profunda para representar relatórios de defeitos por meio de *embeddings*. Nossa abordagem constrói e aprimora a ideia inicial de Deshmukh et al. (2017) em três frentes: (1) uma função de perda aprimorada para informações sobre clusters de relatórios duplicados; (2) representação baseada em atenção de recursos textuais para melhor representar títulos e descrições de relatórios de defeitos e (3) inclusão de informações baseadas em modelagem de tópicos para aumentar a representação textual do relatório, o que de acordo com Nguyen et al. (2012), pode melhorar a detecção de duplicatas adicionando informações globais sobre os relatórios.

Assim, o objetivo do nosso modelo é aprender representações de relatórios de defeitos extraindo representações semânticas baseados em contexto do título, descrição e recursos categóricos não apenas do relatório, âncora e dos exemplos positivos e negativos, mas também de uma representação agregada de seus grupos duplicados (grupos positivos e negativos). Na figura 25, essas informações são mostradas como centroides C_P e C_N, respectivamente, a representação agregada das duplicatas da âncora e das duplicatas do exemplo negativo escolhido aleatoriamente.

Dessa maneira, na Figura 25 apresentamos em alto nível nossa abordagem, composta por cinco componentes: (i) entrada, (ii) modelo, (iii) embedding aprendido, (iv) função de perda e (v) saída.

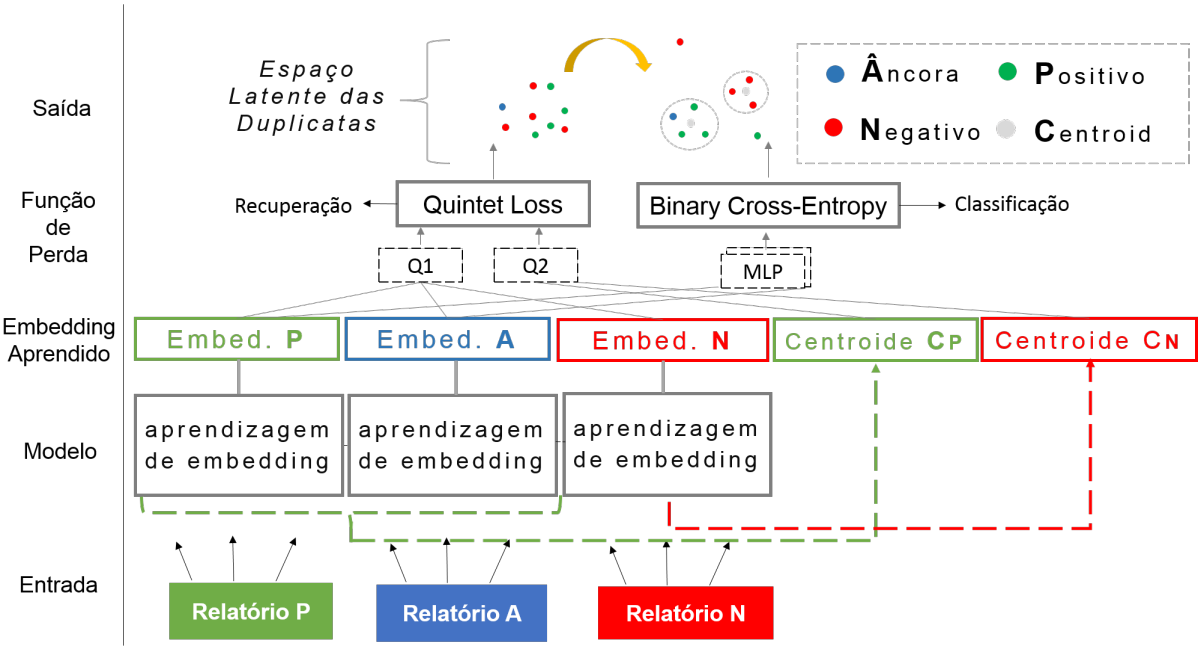


Figura 25 – Arquitetura siamesa profunda para aprender representações sobre relatórios duplicados, combinando exemplos de quintetos, usando uma âncora, um positivo, um negativo e seus centroides.

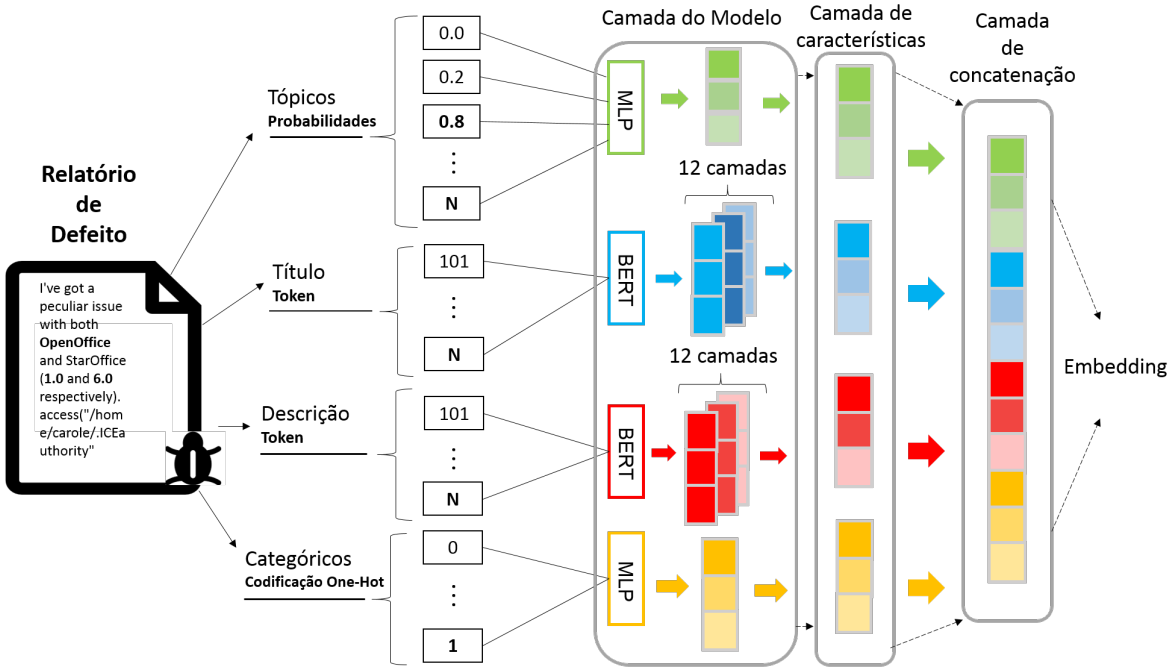


Figura 26 – Aprendizagem de padrões sobre relatórios de defeitos para extrair representações semânticas baseada em contexto.

(I) **Entrada**: cada instância na camada de entrada recebe três relatórios de erros: a âncora, um exemplo positivo e um negativo, representados por seus títulos, descrições, distribuições de tópicos extraídas desses dois

campos de texto e informações categóricas (codificadas como *one-hot*). Nesse componente, SiameseQAT recebe como entrada um quinteto (A, P, N, C_P, C_N), em que A é um relatório âncora, P é um exemplo positivo de defeito duplicado e N um exemplo negativo. Além deses três relatórios, o SiameseQAT também recebe dois vetores *embeddings* que representam clusters de relatórios duplicados C_P e C_N, onde C_P é o *embedding* centroide do cluster de exemplos positivos (o grupo de relatórios aos quais A e P pertencem) e C_N é o centroide do cluster do exemplo negativo N. Um centroide é a média dos vetores *embeddings* de todos os relatórios em um determinado cluster duplicado. Instâncias de exemplo negativo são selecionadas usando o método de tercetos semi-díficil sugerido por Schroff et al. (2015b). Esses negativos, são selecionados muito mais além do positivo, próximos à âncora, porque produzem o melhor treinamento para o modelo de aprendizagem siamesa. Desse modo, empregamos a estratégia de tercetos semi-díficil usando a equação $DIST\hat{A}NCIA(f(A), f(P)) < DIST\hat{A}NCIA(f(A), f(N)) < DIST\hat{A}NCIA(f(A), f(P)) + margem$, uma regra aplicada para seleção dos melhores exemplos negativos para treinamento.

(II) **Modelo**: esse componente é responsável por construir representações semânticas baseadas em contexto para documentos de relatórios de defeitos, com o objetivo de criar representações latentes que conectem as entradas coletadas pelo componente (i) ao modelo que aprenderá *embeddings* sobre os relatórios. Às três caixas marcadas como *aprendizagem de embeddings* na Figura 25, são os modelos de aprendizado profundo formados por duas classes de redes neurais, BERT e MLP. A primeira para o processamento de recursos de texto como *short_desc* e *description* separadamente, conforme feito em Deshmukh et al. (2017), exceto a substituição de LSTM e CNN pelas redes BERT, devido às motivações anteriores discutidas na introdução dessa pesquisa. As camadas MLP são usadas para processar dois tipos de informações: recursos categóricos (produto, componente, prioridade, severidade, resolução, *status*, e versão) e distribuição de tópicos. Assim, agrupando quatro redes, conforme ilustrado na Figura 26, temos um conjunto de redes: BERT no título, BERT na

descrição, MLP nos tópicos e MLP nos categóricos. A saída desses quatro conjuntos é então combinada por uma camada de concatenação, que é o *embedding* do referido relatório. A arquitetura na Figura 25 é uma rede neural siamesa, onde um único conjunto de pesos é usado para aprender a representação latente dos 3 relatórios de defeitos na entrada, que é uma abordagem comum para detecção duplicata usando redes neurais profundas (DESHMUKH et al., 2017).

(III) **Embedding aprendido**: este componente é uma concatenação das quatro representações *embeddings* aprendidas pelo modelo descrito na Figura 26. Em SiameseQAT, o *embedding* de cada componente tem o mesmo tamanho D portanto, a representação final do *embedding* de um relatório de defeito é um vetor com tamanho $4D$. Esse vetor pode ser usado para tarefas de recuperação e classificação.

(IV) **Função de perda**: este componente aplica duas funções matemáticas: a *Quintet Loss* e a entropia cruzada binária, que são treinadas em momentos diferentes. A *Quintet Loss* visa obter a semelhança entre os vetores *embeddings* de relatórios de defeitos da âncora, o exemplo positivo/negativo e seus respectivos centroides gerados pelo componente (iii), seu objetivo é minimizar a distância entre o relatório âncora e seus exemplos positivos, enquanto maximiza a distância para os negativos. Os detalhes sobre as motivações e efeitos do uso da *Quintet Loss* serão explicados na seção 4.4.1. A entropia cruzada binária visa minimizar o erro entre o valor previsto e o valor real para a classificação binária de pares de relatórios duplicados.

(V) **Saída**: a saída do SiameseQAT é o *embedding* de relatórios de defeitos como vetores em um espaço de característica latente. À medida que o treinamento avança, esperamos que os relatórios duplicados terminem mais perto do que os relatórios não duplicados.

Na próxima seção é definido a função *Quintet Loss*, descrevendo seu impacto no aprendizado, além de motivar o uso em nossa solução proposta.

4.4.1 Quintet Loss (QL): Função de Perda Baseada em Informações Coletivas de Duplicatas

Apesar da função *Triplet Loss (TL)* permanecer amplamente usada nas arquiteturas siamesas para aprender a semelhanças entre *embeddings* em modelos de aprendizagem profunda, a mesma também pode ser aplicada para detectar relatórios de defeitos duplicados (DESHMUKH et al., 2017). A função TL foi introduzida por Chechik et al. (2010) e Schroff et al. (2015b) com o objetivo de transformar o espaço latente gerado por um modelo de aprendizado profundo em *clusters* com instâncias semelhantes (SCHROFF et al., 2015a). Todavia, essa função não considera uma distribuição global de dados, pois o treinamento sobre todos os tercetos possíveis é caro e geralmente é empregado por meio de amostragem aleatória (GE, 2018). Além disso, todos os tercetos são tratados igualmente com uma margem constantemente violada, o que motiva abordagens para investigar a reponderação das amostras no treinamento (WU et al., 2017). Pensando nisso, decidimos melhorar a função TL, introduzindo uma nova perda, a *Quintet Loss*, que inclui além das informações sobre grupos de duplicatas juntamente com exemplos individuais, também as informações relativas aos agrupamentos (*clusters*) de relatórios duplicados dos exemplos positivos e negativos, através de seus centróides, visando integrar informações coletivas sobre as duplicatas, para mitigar a importância igual entre os tercetos no treinamento, além de ponderar o treino com informação de contexto por meio dos centroides.

A ideia por trás da Quintet Loss é dupla: (i) evitar igual importância no treinamento original em tercetos, adicionando centroides e novos pesos nas funções de perda, e também (ii) para fazer o modelo gerar *clusters* mais bem definidos e discriminados, adicionando informações sobre todas as réplicas da âncora presentes no lote (o centroide). Na Figura 27, ilustramos o impacto da função no espaço latente das duplicatas, clusterizando as duplicatas com base em seus centroides.

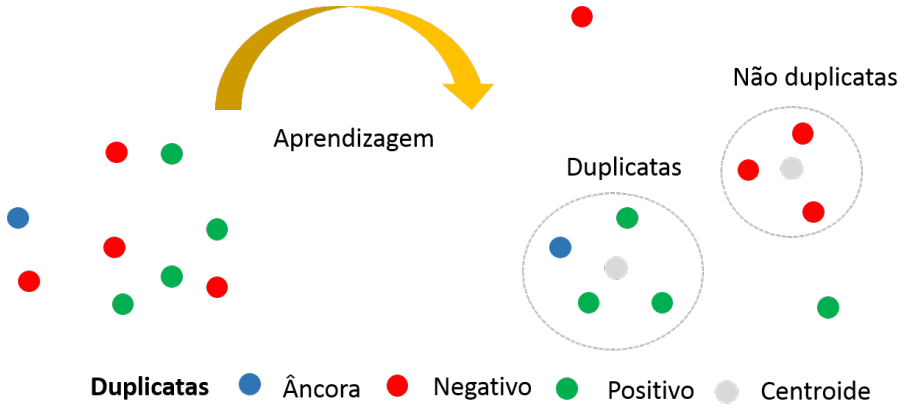


Figura 27 – Ilustração do efeito causado pela Quintet Loss no espaço de características das duplicatas durante o treinamento.

Com isso, esperamos melhorar o aprendizado das instâncias em uma TL, com o objetivo de evitar tercetos com igual influência na fase de treinamento, através de novos pesos e um sistema de equações compostas para tercetos, tentando reduzir o problema de igual importância nas instâncias de tercetos descritas por Wu et al. (2017). Assim, na *Quintet Loss* calculamos dois componentes, 4.7 e 4.8, que são combinados na nova função de perda 4.9.

Nas equações 4.7, 4.8, 4.9, a constante C é um valor definido como 1, aplicando a margem de distância entre os valores de cosseno para positivos e casos negativos, um valor livremente escolhido de acordo com a tarefa. Além disso, a instância A é um relatório de defeito rotulado como âncora, P é uma duplicata de A, N é um relatório de defeito que não é uma duplicata de A. Então, C_P é uma centroide a partir de todas as duplicatas de P (incluindo ela mesma). Portanto, o centroide é o vetor médio das duplicatas. Da mesma forma, C_N é uma centroide de todas as duplicatas de N no lote. A equação Q_1 é a *Triplet Loss* para A, P e N. A equação Q_2 maximiza a distância entre a âncora A e o centroide das duplicatas positivas C_P, enquanto minimiza entre o centroide do C_N negativo. Em nosso método proposto, usamos a semelhança de cosseno entre os relatórios de defeitos como métrica de similaridade.

A *similaridade do cosseno* chamada por $Cosseno$ na fórmula da *Quintet Loss* é definida na equação 4.6, em que A é o vetor de consulta e B é o vetor que é comparado. A função $Coseno_{Distância}$ na equação 4.5 mensura a distância do cosseno entre os vetores A e B que são representados por uma função f. Assim, a distância entre os vetores é encontrada a partir da similaridade

através do produto interno, resultando em valores com faixas contínuas entre -1 quando os vetores são opostos e 1 quando são idênticos. Um valor zero indica ortogonalidade ou descorrelação.

$$COSSENO_{Distância}(f(A), f(B)) = \frac{\sum_{i=1}^{N} f(A_i) f(B_i)}{\sqrt{\sum_{i=1}^{N} f(A_i)^2} \sqrt{\sum_{i=1}^{N} f(B_i)^2}} \tag{4.5}$$

$$COSSENO(f(A), f(B)) = 1 - COSSENO_{Distância}(f(A), f(B)) \tag{4.6}$$

$$Q_1 = MAX(0, C + COSSENO(f(A), f(P)) - COSSENO(f(A), f(N))) \tag{4.7}$$

$$Q_2 = MAX(0, C + COSSENO(f(A), f(C_P)) - COSSENO(f(A), f(C_N))) \tag{4.8}$$

$$L_{quintet} = \frac{(Q_1.W_1 + Q_2.W_2)}{(W_1 + W_2)} \tag{4.9}$$

A função de perda geral em 4.9 é a média ponderada entre Q_1 e Q_2. Neste trabalho, propomos duas alternativas para definir esses pesos. Nossa primeira abordagem é simplesmente usar os mesmos pesos para os dois componentes, transformando a Quintet Loss em uma média simples dos dois componentes. A segunda abordagem é adicionar esses dois pesos como parâmetros treináveis na SiameseQAT e deixar o processo de treinamento escolher os pesos mais adequados para o conjunto de dados de treinamento.

Um cuidado durante o uso da *Quintet Loss* é a explosão dos gradientes, que ocasiona valores extremos para o treinamento, como valores zero ou o maior valor numérico computacionalmente, que caracteriza o fim do aprendizado e da generalização para o modelo. Esse fato, gera uma rede super ou pouco ajustada, devido aos valores absolutos como resultado da similaridade vetorial dos relatórios. Outra recomendação, é uso de transformações não lineares em camadas ocultas que usem a *Quintet Loss*, pois podem ocasionar a explosão dos gradientes. Funções que entreguem gradientes válidos na maioria das vezes, são as mais recomendadas, como a Tangente Hiperbólica (TanH), que

opera sobre valores no intervalo entre -1 e 1, diferente da *Relu*, que transforma valores negativos sempre em 0. A *Relu*, metade das vezes resultará em perda de informação durante o treinamento, entregando 50% das vezes valores inválidos (0 similaridade) para seus gradientes (MAAS; HANNUN; NG, 2013).

No capítulo 5, as abordagens adotadas nessa pesquisa como avaliação para detecção de duplicatas são descritas através de experimentos e análises.

4.5 Considerações finais

Neste capítulo foram apresentadas cada etapa da abordagem de detecção para relatórios duplicados, descrevendo todos os passos, desafios, e uma solução para o problema supracitado. Apesar da proposta mostrar uma estrutura completa para detectar duplicatas, o modelo SiameseQAT consiste na parte principal desta pesquisa, visto que a solução apresentada idealiza *embeddings* que podem ser usados na tarefa de detecção, busca e outras tarefas. Portanto, esperamos construir representações significativas sobre relatórios de defeitos, extraindo informações semânticas e contextuais sobre os documentos, aprendendo a clusterizar as duplicatas no espaço latente de características.

5 Experimentos e Avaliação dos Resultados

Neste capítulo, apresentamos os experimentos realizados para avaliar o impacto do SiameseQAT na recuperação e classificação de relatórios de defeitos duplicados em comparação com nossos *baselines*. Também apresentamos uma análise qualitativa das representações *embeddings* construídas para os relatórios pelo modelo.

A nossa abordagem e os *baselines* foram implementados usando o Keras (CHOLLET et al., 2015), uma biblioteca de alto nível para experimentos usando aprendizado profundo, que possui código aberto, e foi escrita em Python. Todos os eperimentos foram realizadas em um servidor, *Intel i7-7700* com 64GB de RAM e duas placas de vídeo *Nvidia GeForce GTX 1080 Ti*. Todo o código-fonte e conjuntos de dados estão disponíveis em um repositório público[1] para fins de reprodutibilidade.

Este capítulo está dividido da seguinte maneira. Na Seção 5.1, apresentamos as principais características das bases de treinamento. Na Seção 5.2, descrevemos os baselines em comparação a nossa proposta. Na Seção 5.3, apresentamos as configurações para a fase de experimentação. Na Seção 5.4 e 5.5, apresentamos os resultados para duas abordagens de detecção das duplicatas, a recuperação e classificação. Na Seção 5.6, discutimos sobre o impacto gerado pelo SiameseQAT no espaço latente das duplicatas. Nesse sentido, na Seção 5.7, fornecemos também uma avaliação qualitativa do impacto gerado pelo SiameseQAT em sequências textuais usando mecanismos de atenção por meio da BERT.

5.1 Características da Base de Treinamento

Nos experimentos foram usados três repositórios de relatórios de defeitos de grandes projetos de código aberto para a tarefa de detecção das duplicatas: Eclipse, NetBeans e Open Office, coletados e publicados por Lazar, Ritchey

[1] https://github.com/thiagomarquesrocha/siameseQAT

e Sharif (2014), e separamos as principais características observadas para cada conjunto de dados. Dentre elas: estatísticas auxiliares sobre as bases, uma avaliação qualitativa sobre os tópicos de palavras extraídos de cada base, além das características espaciais dos atributos categóricos nos relatórios de defeitos.

Na Tabela 9, demonstramos a proporção das duplicatas, e outras informações sobre os atributos textuais e categóricos. Sobre os atributos categóricos, na Tabela 10, podemos verificar que a soma de todos os valores distintos para os campos categóricos é esparso, devido ao alto número de componentes, versões e produtos em relatórios de defeitos para às três bases de dados.

Tabela 9 – Estatísticas sobre as bases de dados.

		Eclipse	**NetBeans**	**Open Office**
Duplicatas	Nº duplicatas	233.415	248.666	132.518
	Nº de relatórios únicos	271.901	162.921	52.440
Grupos de Duplicatas	Nº de grupos	62.521	53.794	19.794
	Nº de grupos distintos (tamanho)	35	43	38
	Menor grupo de duplicatas	2	2	2
	Tamanho do grupo no 1º quartil	10	12	11
	Tamanho médio de grupos	19	23	20
	Tamanho do grupo no 3º quartil	28	36	29
	Maior grupo de duplicatas	50	56	99
Textual	Nº de palavras distintas *	20.000	19.061	18.562
	Tamanho médio de *title*	15	16	13
	Tamanho médio de *description*	326	796	207
Categórico	Nº de *bug_severity* distintos	7	7	6
	Nº de *bug_status* distintos	3	3	3
	Nº de *component* distintos	908	473	137
	Nº de *priority* distintos	5	4	5
	Nº de *product* distintos	189	39	41
	Nº de *version* distintos	547	18	537

* Tamanho do vocabulário restrito às 20.000 palavras mais frequentes devido a restrições de memória.

Tabela 10 – Quantidade de valores distintos para campos categóricos nas bases de dados.

	Eclipse	**NetBeans**	**Open Office**
Total de valores categóricos distintos	1.659	544	729

No Capítulo 2 foi demonstrado como visualizar os relatórios de defeitos na visão de tópicos de palavras. Desse modo, foram extraídos 30 tópicos para cada base de treinamento, permitindo analisar o contexto dos relatórios através das palavras-chave a respeito dos defeitos de *software*. Assim, na base do Open Office, por exemplo, analisando as top 15 palavras no tópico 4, encontramos vocabulários dos tipos: *install, user, error, file, run, crash, office, try, use, message, installation, instal, problem, work, start*. Um tópico aparentemente descrevendo sobre erros durante a instalação e uso da ferramenta Open office.

Para a base do Eclipse, analisando as top 15 palavras nos tópicos 23 e 26, encontramos vocabulários dos tipos: *eclipse, org, java, run ,core, dget, face, text, ger, mana, work, ion, act, lang, method, lass, load, java, bundle, del, core, , aspect, start, bund, ete e ada* — tópicos que descrevem componentes técnicos extraídos de duplicatas com *stack traces*, código-fonte, passos para reproduzir e *logs*, muito característico de bases como a Eclipse e NetBeans.

5.2 Baselines

Comparamos o SiameseQAT com *baselines* no estado-da-arte baseados em aprendizagem profunda (AP) para detectar relatórios de defeitos duplicados. Nossa comparação focou em abordagens baseadas em AP devido a Budhiraja et al. (2018a) terem demonstrado que seus resultados foram significativamente melhores em comparação com os métodos baseados em recuperação de informação anteriores. Além disso, é importante observar que nenhum dos esforços anteriores baseados em AP possuem comparação direta usando os mesmos dados de treinamento ou referência entre si. No entanto, nossa abordagem apresentará uma comparação direta entre cada um. Os *baselines* escolhidos foram:

- **DWEN**: *Deep Word Embedding Neural Network* (DWEN) proposto por Budhiraja et al. (2018a) e Budhiraja et al. (2018b). A abordagem foi modelada para classificar um par de relatórios de defeitos como duplicados ou não, considerando apenas recursos textuais em um modelo baseado no método Skip-Gram. O modelo DWEN gera um vetor *embedding* para o relatório e

o candidato, codificando o par em uma função *Sigmoid* para gerar como saída a probabilidade de ser duplicado. A abordagem original Budhiraja et al. (2018a) não avalia o modelo para tarefas de classificação. No entanto, em Budhiraja et al. (2018b), os autores estenderam a pesquisa para incluir experimentos de classificação. Assim, em nossos experimentos, usamos a camada *Sigmoid* de DWEN para classificação binária, e a camada anterior à Sigmoid para codificar os relatórios de defeitos de todos os conjuntos de dados, empregando as tarefas de recuperação. Dessa maneira, comparando o DWEN com o SiameseQAT, o método DWEN não extrai nem usa tópicos de texto dos conjuntos de dados, não emprega mecanismos de atenção e desconsidera os desafios dos relatórios de defeitos como *stack traces*, *logs* e código-fonte no conteúdo textual. Além disso, eles não usaram recursos categóricos.

- **DMS**: *Deep Siamese Model* (DMS) proposto por Deshmukh et al. (2017), constrói um vetor especial criado pela combinação de título, descrição e categóricos (produto, componente, prioridade, severidade, resolução, *status*, e versão) dos relatórios de defeitos. Esse *baseline*, usa três tipos de camadas neurais: uma MLP, uma CNN, e uma BiLSTM para codificar relatórios de defeitos. Comparada a nosso trabalho, a proposta DMS corresponde à abordagem mais semelhante a nossa, pois além de combinar diferentes redes neurais, também utilizam a arquitetura siamesa para processar relatórios duplicados, recebendo como entrada recursos textuais e categóricos. A avaliação do DMS contempla tanto tarefas de classificação quanto em recuperação das duplicatas, o que também iremos avaliar. No entanto, DMS usa redes diferentes da nossa para textos, e em sua abordagem não consideram o uso de mecanismos de atenção no aprendizado, além de não incluírem uma representação baseada em tópicos para os textos. Em nossa abordagem, formamos uma arquitetura profunda para entender os recursos de modelagem para tópicos e combinamos com o aprendizado baseado na atenção para formar uma representação baseada em contexto semântico para os recursos de relatórios.

Em todas as implementações dos *baselines*, empregamos as mesmas

configurações e parâmetros referente a arquitetura dos modelos, conforme descrito nos respectivos trabalhos. No entanto, em relação a hiperparâmetros como o número épocas, treinamos os *baselines* e a proposta desse trabalho sob as mesmas condições, modificando as configurações original dos *baselines*. Portanto, todos esses detalhes serão descritos na próxima seção.

5.3 Treinamento e Hiperparâmetros

O treinamento de todos os modelos, *baseline* e proposta, foram feitos em 1000 épocas inicialmente, para treinar a fase de recuperação das duplicatas. Nesse sentido, observamos a curva de perda no treino e teste identificando quando as curvas passam a convergir em um determinado número de épocas. Desse modo, quando às curvas se encontram, sabe-se que é o ponto ideal de parada para o treino. Portanto, após esse treino, configuramos os modelos para serem treinados por mais 100 épocas para uma nova fase de detecção, a classificação das duplicatas. Assim, em todas as fases, o mesmo conjunto separado para treino, validação e teste é utilizado para coletar os resultados da avaliação.

Para entender melhor as contribuições individuais de nossa proposta, realizamos experimentos com diversas variações do SiameseQAT. O SiameseQAT é o nosso método completo proposto, incluindo recursos com base em mecanismos de atenção, tópicos e informações de agrupamento, usando o *Quintet Loss* como sua função de perda. Também avaliamos o impacto dos recursos baseados em tópicos. As variações sem tópicos tiveram a última letra *T* removida, sendo SiameseQA a versão com textitQuintet Loss e SiameseTA a versão com *Triplet Loss*. Além disso, avaliamos duas configurações para definir os pesos da perda de quinteto. Ao usar pesos fixos com valores idênticos, adicionamos o sufixo-A ao nome, como no *SiameseQA-A*. Da mesma forma, ao usar pesos treináveis, adicionamos o sufixo-W ao nome. Por padrão, quando nos referirmos sem o sufixo, assuma que é a versão com pesos fixos.

Os relatórios de defeitos são passados como entradas para a solução da seguinte maneira: *textual* representa o título e descrição; *categórico* os campos, produto, componente, prioridade, severidade, resolução, *status*, e versão, os *tópicos* são representações distribuídas em tópicos sobre o vocabulário textual.

Dessa maneira, avaliamos a solução proposta identificando se os ganhos percebidos são devidos à implantação da BERT para representação textual, ou à influência do tópico como um novo recurso, ou também à implantação da função de perda *Quintet Loss*. Portanto, detalhamos as configurações das propostas na Tabela 11. Os experimentos foram avaliados em duas tarefas de detecção das duplicatas, recuperação e classificação.

Para a tarefa de recuperação, usamos a camada *modelo* na arquitetura da Figura 26 para codificar novos relatórios de defeitos. Para a tarefa de classificação, usamos as mesmas camadas presentes no modelo de recuperação e transferimos seus pesos pré-treinados enquanto os congelamos para modificações posteriores. Em seguida, adicionamos duas camadas MLP com 64 dimensões, usando *Softmax* como sua saída e entropia cruzada binária como a função de perda. Todos os modelos e *baselines* foram treinados usando ADAM (KINGMA; BA, 2014) como otimizador, por causa do rápido desempenho em comparação a outros otimizadores como SGD, além da pequena tunagem de parâmetros que ocorre em seu mecanismo de aprendizado no ADAM.

Tabela 11 – Configuração de características para todos os modelos de aprendizagem profunda, incluindo os *baselines* e o proposto.

Configuração	**DMS**	**DWEN**	**SiameseTAT**	**SiameseQAT**
Mecanismo de Atenção			X	X
Função *Quintet*				X
Função *Triplet*	X		X	X
Pesos fixos na *loss*				X
Pesos treináveis na *loss*				X
Função *Cross Entropy*		X	X	X
Ativação *Tanh*	X	X	X	X
Ativação *Sigmoid*		X		
Ativação *Softmax*			X	X
Pré-treino *Glove*	X	X		
Pré-treino BERT			X	X
Rede MLP	X	X	X	X
Rede BERT			X	X
Rede CNN	X			
Rede BiLSTM	X			
Rede siamesa	X		X	X
Entrada com quintetos				X
Entrada com tercetos	X		X	X
Entrada textual	X	X	X	X
Entrada categórica	X		X	X
Entrada de tópicos			X	X

Em relação aos modelos pré-treinados, para a nossa abordagem, usamos o dicionário pré-treinado da BERT para fazer transferência de aprendizagem, enquanto para o treinamento dos modelos de Deshmukh et al. (2017) e Budhiraja et al. (2018a) o *Glove* foi utilizado.

Em nosso método SiameseQAT temos dois conjuntos de camadas BERT e dois conjuntos MLP, com o *embedding* de relatórios de defeitos representados em um vetor com 1200 dimensões, onde cada 300 dimensões são para cada conjunto de recursos, como mostra a Figura 26. Utilizamos 100 palavras como tamanho da frase para cada recurso de texto, devido a limitados recursos computacionais, mas foram avaliados diferentes tamanhos para o total de palavras como 20, 30 e 50. Para tópicos de palavras, extraímos 30 tópicos para cada documento de relatório de defeito. As camadas do BERT foram pré-treinadas com o modelo *'uncased_L-12_H-768_A-12'*, que possui pesos para um bilhão de tokens pré-treinados em um grande corpus[2]. As camadas MLP

[2] https://github.com/google-pesquisa/bert

têm 600 unidades totalmente conectadas usando a tangente hiperbólica como sua função de ativação.

Para a BERT, usamos 12 camadas presentes na arquitetura de Devlin et al. (2018), descongelando 8 camadas para treinamento. Essa arquitetura adotada é inspirada nas recomendações de Jain e Wallace (2019), que esclarecem o uso do BERT como um extrator de recursos para codificar sequências textuais e gerar representação semântica. A aplicação da BERT oferece benefícios do pré-treinamento em vários idiomas, além de conceder a manipulação aritmética de suas camadas de saída (como soma, média ou concatenação). Essa manipulação visa capturar distintas peculiaridades aprendidas nas entradas de texto.

5.4 Recuperação de Duplicatas

Nesta abordagem avaliamos todos os modelos como se fossem um problema de busca, onde esperamos que para cada relatório X dado como consulta, uma lista de k relatórios candidatos sejam recuperados como possíveis candidatas às duplicatas. Assim, confirmamos se uma lista com K relatórios contém pelo menos uma duplicata, caracterizando como 100% de acurácia para a consulta. No final, a avaliação calcula uma média de todas as consultas, nomeada pela literatura como *Recall@k* (BUDHIRAJA et al., 2018a; SUN et al., 2011). Para cada consulta, testamos relatórios duplicados nunca vistos pelo modelo durante o treinamento.

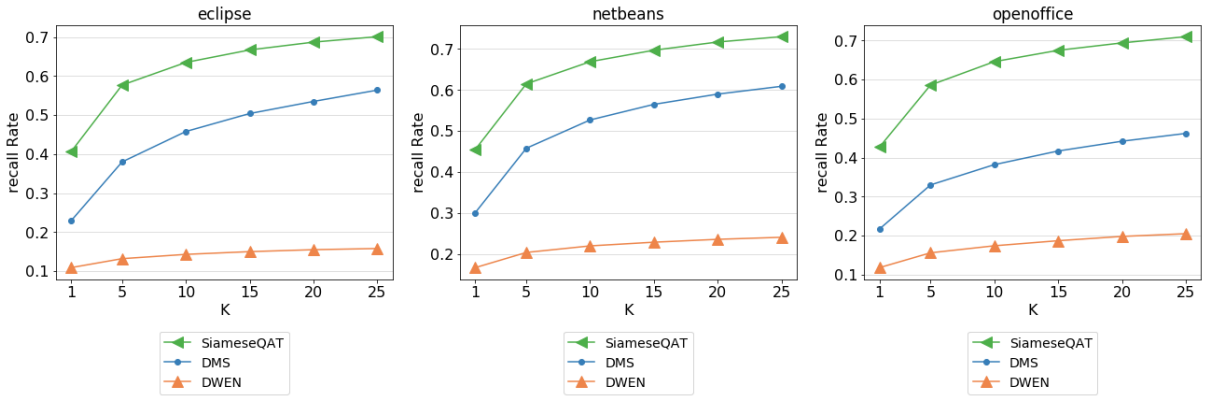


Figura 28 – Recall@K médio para vários valores de K e todos os *baselines* em comparação ao SiameseQAT.

Tabela 12 – Recall@K médio para Eclipse e todas as abordagens.

Eclipse						
	R@1	R@5	R@10	R@15	R@20	R@25
DWEN	0.11	0.13	0.14	0.15	0.16	0.16
DMS	0.23	0.38	0.46	0.50	0.54	0.56
SiameseTA	0.37	0.53	0.59	0.62	0.64	0.65
SiameseTAT	0.39	0.57	0.63	0.66	0.68	0.69
SiameseQA-W	0.37	0.53	0.59	0.62	0.64	0.65
SiameseQA-A	0.38	0.54	0.60	0.63	0.65	0.67
SiameseQAT-W	0.40	0.57	0.63	0.66	0.68	0.69
SiameseQAT-A	**0.41**	**0.58**	**0.64**	**0.67**	**0.69**	**0.70**

Tabela 13 – Recall@K médio para NetBeans e todas as abordagens.

NetBeans						
	R@1	R@5	R@10	R@15	R@20	R@25
DWEN	0.17	0.20	0.22	0.23	0.24	0.24
DMS	0.30	0.46	0.53	0.57	0.59	0.61
SiameseTA	0.42	0.58	0.63	0.66	0.68	0.69
SiameseTAT	0.44	0.60	0.66	0.69	0.71	0.72
SiameseQA-W	0.43	0.58	0.63	0.66	0.68	0.69
SiameseQA-A	0.43	0.59	0.64	0.67	0.69	0.70
SiameseQAT-W	0.45	0.61	**0.67**	0.69	0.71	**0.73**
SiameseQAT-A	**0.46**	**0.62**	**0.67**	**0.70**	**0.72**	**0.73**

Tabela 14 – Recall@K médio para Open Office e todas as abordagens.

Open Office						
	R@1	R@5	R@10	R@15	R@20	R@25
DWEN	0.12	0.16	0.17	0.19	0.20	0.21
DMS	0.22	0.33	0.38	0.42	0.44	0.46
SiameseTA	**0.46**	0.61	0.67	0.69	0.71	0.73
SiameseTAT	**0.46**	**0.62**	**0.68**	**0.71**	**0.73**	**0.74**
SiameseQA-W	0.41	0.57	0.63	0.66	0.67	0.69
SiameseQA-A	0.42	0.58	0.63	0.66	0.68	0.70
SiameseQAT-W	0.44	0.60	0.65	0.68	0.70	0.72
SiameseQAT-A	0.43	0.59	0.65	0.68	0.69	0.71

A Figura 28 mostra os resultados obtidos pela melhor variação de SiameseQAT em comparação com os *baselines* em termos de *Recall@k*. Como pode

Tabela 15 – Avaliação do ganho relativo estatístico entre os valores de recall@k no top 5 usando o test-t para o Open Office em todas as abordagens.

Open Office								
	DMS	DWEN	STA	STAT	SQAW	SQAA	SQATW	SQATA
STA	84.8%	281.2%	-	**-1.6%**	7.0%	5.1%	1.6%	3.3%
STAT	**87.8%**	**287.5%**	**1.6%**	-	**8.7%**	**6.8%**	**3.3%**	**5.0%**
SQAW	72.7%	256.2%	-6.5%	-8.0%	-	-1.7%	-5.0%	-3.3%
SQAA	75.7%	262.5%	-4.9%	-6.5%	1.7%	-	-3.3%	-1.6%
SQATW	81.8%	275.0%	-1.6%	-3.2%	5.2%	3.4%	-	1.6%
SQATA	78.7%	268.7%	-3.2%	-4.8%	3.5%	1.7%	-1.6%	-

Em negrito o(a)s maiores ganhos/perdas.
Sublinhado representa os resultados estatisticamente significativos.

Tabela 16 – Avaliação do ganho relativo estatístico entre os valores de recall@k no top 5 usando o test-t para o NetBeans em todas as abordagens.

NetBeans								
	DMS	DWEN	STA	STAT	SQAW	SQAA	SQATW	SQATA
STA	26.0%	190.0%	-	-3.3%	0%	-1.6%	-4.9%	-6.4%
STAT	30.4%	200.0%	3.4%	-	3.4%	1.6%	-1.6%	-3.2%
SQAW	26.0%	190.0%	0%	-3.3%	-	-1.6%	-4.9%	-6.5%
SQAA	28.2%	195.0%	1.7%	-1.7%	1.7%	-	-3.2%	-4.8%
SQATW	32.6%	205.0%	5.1%	1.6%	5.1%	3.3%	-	**-1.6%**
SQATA	**34.7%**	**210.0%**	**6.8%**	**3.3%**	**6.8%**	**5.0%**	**1.6%**	-

Em negrito o(a)s maiores ganhos/perdas.
Sublinhado representa os resultados estatisticamente significativos.

Tabela 17 – Avaliação do ganho relativo estatístico entre os valores de recall@k no top 5 usando o test-t para o Eclipse em todas as abordagens.

Eclipse								
	DMS	DWEN	STA	STAT	SQAW	SQAA	SQATW	SQATA
STA	39.4%	307.6%	-	-7.0%	0%	-1.8%	-7.0%	-8.6%
STAT	50.0%	338.4%	7.5%	-	7.5%	5.5%	0%	-1.7%
SQAW	39.4%	307.6%	0%	-7.0%	-	-1.8%	-7.0%	**8.6%**
SQAA	42.1%	315.3%	1.8%	-5.2%	1.8%	-	-5.2%	-6.8%
SQATW	50.0%	338.4%	7.5%	0%	7.5%	5.5%	-	-1.7%
SQATA	**52.6%**	**346.1%**	**9.4%**	**1.7%**	**9.4%**	**7.4%**	**1.7%**	-

Em negrito o(a)s maiores ganhos/perdas.
Sublinhado representa os resultados estatisticamente significativos.

ser visto, o método proposto alcança resultados consistentemente superiores, revelando níveis de revocação entre 12% e 25% melhores. No conjunto de dados Open Office, a melhoria de revocação foi ainda mais acentuada, com um aumento de cerca de 25% quando comparado com o melhor *baseline*, o DMS. Essa é uma forte indicação de que as contribuições propostas neste trabalho tiveram um impacto significativo na separação de relatórios distintos, mantendo as duplicatas próximas. No entanto, é importante entender também qual é a contribuição de cada componente de SiameseQAT nesse aumento visível de revocação.

A tabela 12, 13 e 14 apresentam os resultados alcançados por todas as variações dos componentes de SiameseQAT, bem como os *baselines*, para os três avaliados conjuntos de dados. Ao considerar o recurso de probabilidades do tópico, pode-se observar que em todos os conjuntos de dados, sua inclusão leva a aumentos consistentes de revocação para todos os valores de K considerados, variando de 0.02 e 0.04 no Eclipse e NetBeans. No Open Office, esse aumento foi um pouco menor, variando de nenhum aumento a 0.3.

Ao considerar o impacto do uso da Quintet Loss, nos conjuntos de dados Eclipse e NetBeans usando a média de pesos (SiameseQAT e SiameseQA-A) resultam em valores superiores quando comparados ao uso da perda baseada em tercetos, com ganhos em torno de 0.01 a 0.02 para todos os níveis de revocação. No entanto, no Open Office, o uso da *Triplet Loss* leva a melhores resultados. Acreditamos que essa diferença de níveis de revocação pode ser explicada pelo fato de esse conjunto de dados ter um número muito menor de *clusters* e duplicatas do que os outros dois conjuntos de dados. Isso pode indicar que a *Quintet Loss* é mais adequada para conjuntos de dados com um grande número de duplicatas.

Para melhor entender a comparação entre todos os métodos, as tabelas 15, 16 e 17 mostram uma comparação dos níveis de ganho relativos entre todos os métodos, com o valor sendo o percentual relativo de ganho/perda do método da linha sobre o da coluna em termos de R@5 para os três conjuntos de dados. Além disso, todos os nomes dos métodos foram abreviados para visualizar em uma mesma tabela o comparativo, assim na lista de siglas deste trabalho são descritas cada abreviatura. As tabelas também mostram

se a diferença entre eles foi estatisticamente significante através do teste-t de student considerando p<0.05 como rejeitando a hipótese nula, representado pelo valor estar sublinhado. Apesar de estarmos mostrando os resultados para R@5, os resultados para outros níveis de revocação levam a conclusões similares, especialmente conforme o valor de k aumenta, e serão suprimidos por questão de brevidade.

Algo que é visivel em todas as tabelas é que os ganhos obtidos pelas variações propostas nesta dissertação foram substanciais e estatisticamente significantes em comparação com os baselines. Ao se analisar o resultado para cada base de dados, podemos verificar que nas bases de dados Eclipse e Netbeans, apesar dos valores de ganho serem diferentes, a relação entre os métodos e a significância estatística é similar na maior parte das comparações. Os ganhos de usar apenas a Quintet Loss sem utilizar tópicos foi estatisticamente significante na coleção NetBeans ao usar a média simples de pesos (SQAA), porém o mesmo não aconteceu na base Eclipse, onde o ganho observado não foi estatisticamente significante. Com relação à base de dados Open Office, os ganhos observados anteriormente pelas variações sem o uso da Quintet Loss foram significativos com relação às suas versões usando esta função de perda. Isto é uma forte indicação que esta coleção de dados possui características que não são ótimas para o uso da informação de centróides da forma que foi proposta nesta dissertação. Em trabalhos futuros, pretendemos estudar as razões que levaram a este comportamento nesta base de dados em específico. Ainda assim, é importante salientar que em comparação com os baselines as todas as mudanças de abordagem propostas levaram a ganhos expressivos na tarefa de recuperação.

5.5 Classificação de Duplicatas

Nesta abordagem foram avaliadas versões de classificação de pares de relatórios de defeitos nas coleções Eclipse, Open Office e NetBeans. Para cada par de relatórios, o modelo treinado vetorizando-os e classificando-os como duplicados ou não. Para avaliar essa tarefa, usamos duas métricas, Acurácia e AUROC, além da matriz de confusão para confirmar quantas vezes os modelos

classificam corretamente um par de relatórios duplicados entre as duas classes. Na Tabela 18 é apresentado os resultados para todos os métodos, com os melhores resultados em negrito. Nessa seção, apresentamos as matrizes de confusão geradas para todos os modelos por meio das Figuras 29, 30 e 31.

Tabela 18 – Acurácia e AUROC para classificação binária, classe duplicata (1 - um) e não-duplicata (0 - zero).

	Eclipse		**NetBeans**		**Open Office**	
	Acurácia	AUROC	Acurácia	AUROC	Acurácia	AUROC
DWEN	0.67	0.73	0.71	0.79	0.69	0.76
DMS	0.95	**0.99**	0.94	**0.99**	0.90	0.97
SiameseTA	0.93	0.98	0.91	0.97	0.92	0.98
SiameseTAT	0.93	0.98	**0.95**	**0.99**	**0.94**	0.98
SiameseQA-W	0.94	**0.99**	**0.95**	**0.99**	**0.94**	0.98
SiameseQA-A	0.95	**0.99**	0.94	**0.99**	0.93	0.98
SiameseQAT-W	0.91	0.92	0.91	0.91	0.90	0.90
SiameseQAT-A	**0.96**	**0.99**	**0.95**	**0.99**	**0.94**	**0.99**

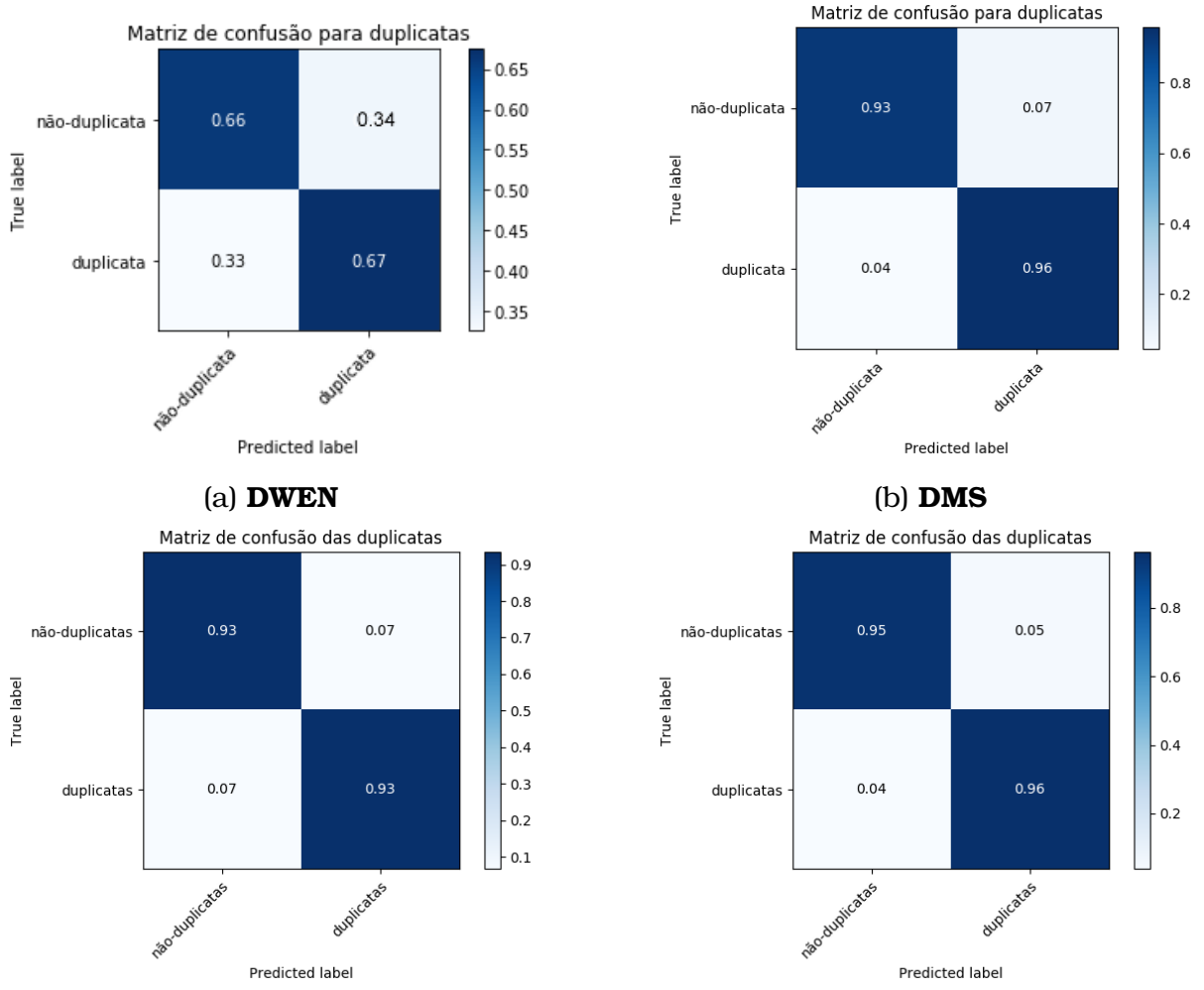
Como pode ser visto, SiameseQAT alcançou os melhores resultados gerais nas duas métricas para todos os conjuntos de dados com margens de diferença variadas em comparação com SiameseTA e *baseline* DMS, estando no máximo tecnicamente empatadas. Quando comparada ao DMS, o melhor *baseline*, SiameseQAT foi superior em termos de precisão em todos os conjuntos de dados, com sua maior melhoria no conjunto de dados Open Office (4,44%). Em termos de AUROC, o aumento foi mais modesto, de 0.97 para 0.99 neste mesmo conjunto de dados, com empates com os outros dois conjuntos de dados de 0.99, um valor que tem um pequeno espaço para melhorias.

Ao comparar os componentes de SiameseQAT, a combinação de todos os componentes levam a resultados consistentemente superiores. O uso de informações do tópico leva a melhorias em todos os cenários, exceto quando associado à *Triplet Loss* em Eclipse. O uso da *Quintet Loss* leva a melhorias maiores no conjunto de dados Eclipse, com uma precisão cerca de 3.2% maior. Nos outros conjuntos de dados, o uso da *Quintet Loss* gera ganhos consistentes somente quando não estiver junto com as informações de tópico. Ao adicionar também tópicos, o resultado foi um empate. Isso pode ser uma

indicação de que há um espaço muito pequeno para melhorias nesses níveis de precisão e, portanto, qualquer um dos componentes pode levar a essa melhoria individualmente. No entanto, essa é uma forte indicação de que o uso dos componentes propostos pode levar a melhores resultados, não apenas nas tarefas de recuperação, mas também nas tarefas de classificação.

Por fim, as matrizes de confusão nas Figuras 29, 30 e 31, confirmam que SiameseQAT é o melhor classificador em comparação aos outros com pequenas margens de erro, demonstrando que o modelo classifica bem as duas classes, tanto duplicatas quanto para não duplicatas.

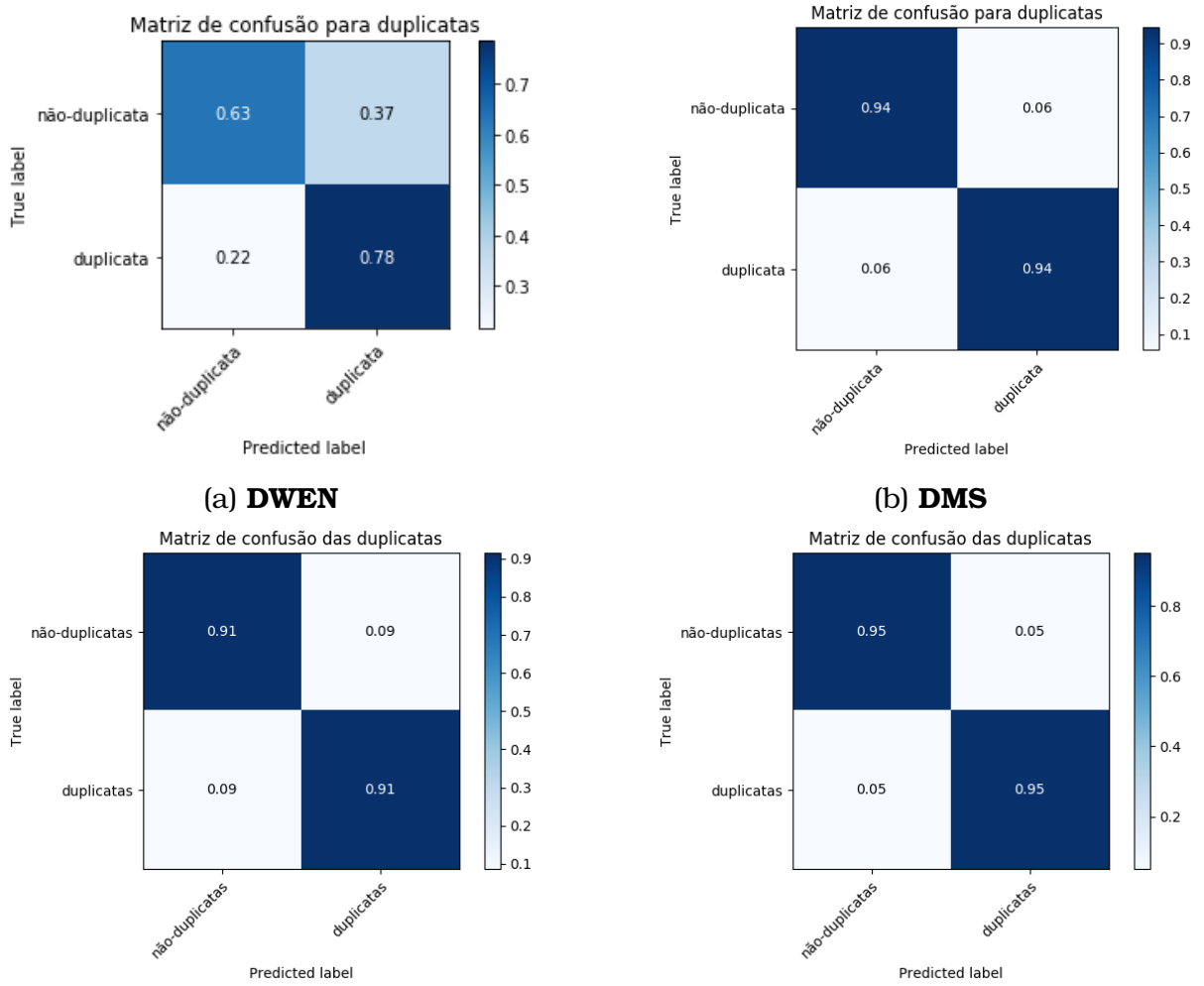
Os resultados para classificação indicam que esta tarefa é diferente da tarefa de recuperação e que uma não deve servir de subsídio para se tirar conclusões sobre a outra. Na seção 5.6, apresentamos algumas hipóteses sobre as causas desse comportamento.



(a) **DWEN** (b) **DMS**

(c) **SiameseTAT** (d) **SiameseQAT**

Figura 29 – Matriz de confusão para Eclipse.



(a) **DWEN** (b) **DMS**

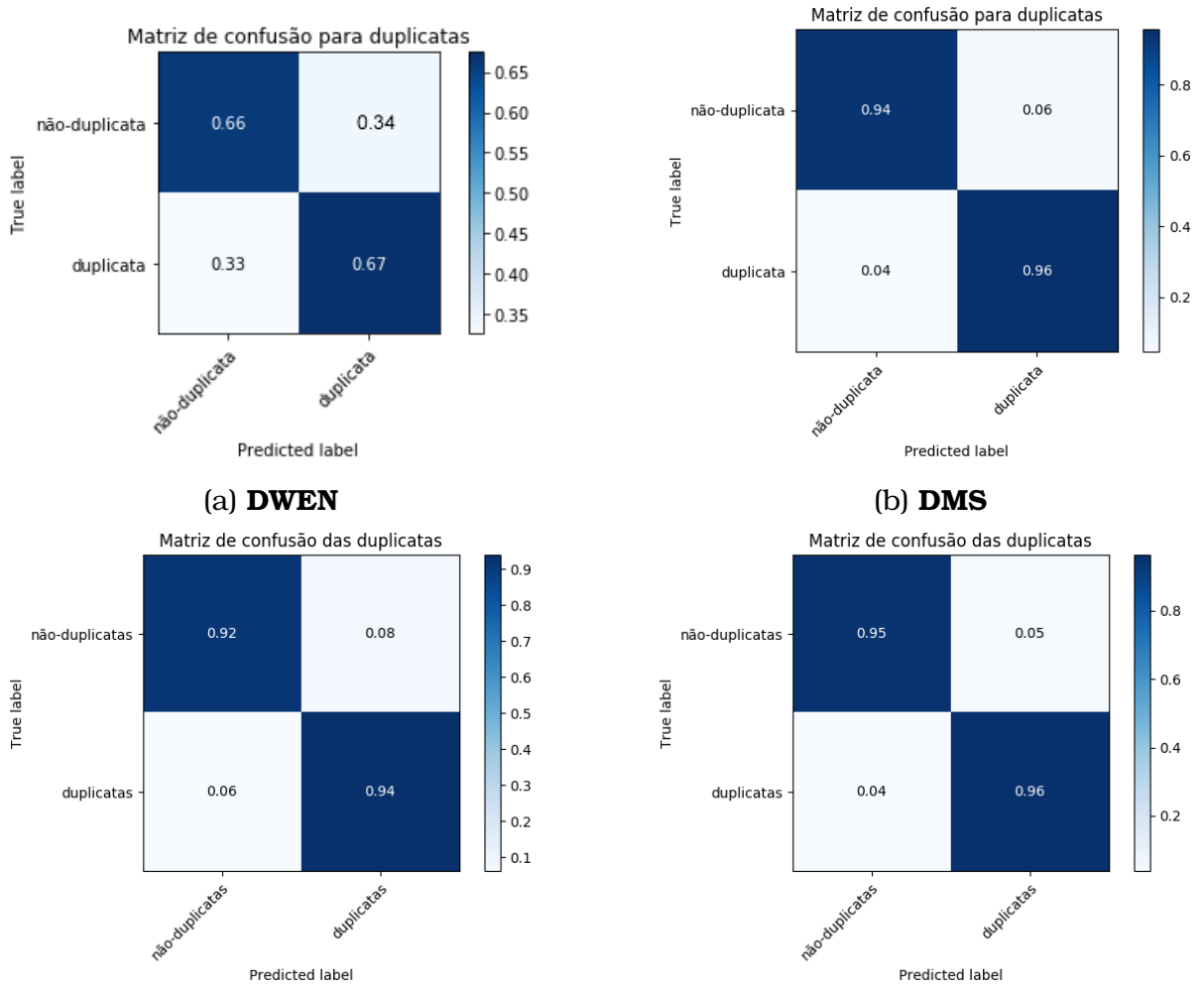
(c) **SiameseTAT** (d) **SiameseQAT**

Figura 30 – Matriz de confusão para NetBeans.

5.6 Agrupamento de Duplicatas

Com o objetivo de visualizar como a *Quintet Loss* e os recursos baseados em contexto semântico organizam os relatórios no espaço latente, selecionamos alguns *clusters* duplicados aleatoriamente para verificar como os diferentes métodos os organizam no espaço. Para cada método, plotamos os *embeddings* de relatórios dos grupos de duplicatas em um espaço bidimensional usando o t-SNE[3], para melhor visualizar como está organizado. Além disso, também exibimos os resultados após 1000 épocas (o final do treinamento). Por uma questão de brevidade, apresentamos essa visualização apenas para o conjunto de dados Open Office, mas um comportamento semelhante foi encontrado nos três conjuntos de dados. Por fim, calculamos uma medida de qualidade para os

[3] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html



(a) **DWEN** (b) **DMS**

(c) **SiameseTAT** (d) **SiameseQAT**

Figura 31 – Matriz de confusão para Open Office.

clusters usando a pontuação *Silhouette* entre os grupos duplicados para todos os conjuntos de dados, mensurando o quão bem nosso modelo produz *clusters*. A Tabela 19 sumariza os resultados.

As configurações utilizadas para o t-SNE que originaram a visualização em duas dimensões sobre os grupos de duplicatas foram três, apesar de existirem inúmeros parâmetros. Nossa seleção focou em configurações sobre iterações e a quantidade de dimensões a ser considerado como otimização. Portanto, os parâmetros adotadas foram:

- **Número de componontes**: O total de 2 componentes para reduzir as D dimensões de cada relatório duplicado dado como entrada. Essa configuração é selecionada para definir em quantas dimensões deve ser reduzido a entrada.

- **Perplexidade**: um valor relacionado ao número de vizinhos mais próximos, e segundo a documentação oficial do Scikit [4], esse valor precisa ser entre 5 e 50, e base de dados grandes, normalmente exigem valores de perplexidade alto. Dessa maneira, adotamos o valor 40 como perplexidade.
- **Número de iterações**: um valor de iterações que o algoritmo irá efetuar para reduzir e otimizar sobre as entradas. Empiricamente adotamos um total de 300 iterações.

Tabela 19 – Avaliação dos grupos de duplicatas no conjunto de teste usando *silhoutte score* para todas as bases.

	Eclipse	**NetBeans**	**Open Office**
	Silhoutte score	Silhoutte score	Silhoutte score
DWEN	-0.36	-0.41	-0.44
DMS	-0.09	-0.09	-0.15
SiameseTA	0.02	0.02	0.03
SiameseTAT	0.02	0.03	**0.03**
SiameseQA-W	0.02	0.03	0.02
SiameseQA-A	0.02	0.03	0.02
SiameseQAT-W	**0.03**	0.03	0.02
SiameseQAT-A	**0.03**	**0.04**	0.02

Na Figura 32c, podemos ver o espaço latente das duplicatas de SiameseQAT, onde, após 1000 épocas, os *clusters* duplicados são bem formados, sem interseção entre eles. Na Figura 32b, mostra o gráfico para SiameseTAT, onde pode ser visto que os clusters estão um pouco misturados, com duplicatas de 19477 e 100934 mostrando possíveis sobreposições. Por fim, a Figura 32a, mostra o mesmo para o DMS, onde não apenas os *clusters* são muito mais misturados, também há a presença de um *outlier* como um exemplo duplicado do grupo 19477 próximo aos duplicados de 28471. Esses resultados podem ajudar a explicar a precisão da classificação do SiameseQAT em comparação com os outros, uma vez que possui *clusters* mais bem formados, o que facilitaria encontrar um hiperplano de separação entre um relatório e suas não duplicatas. Além disso, a presença desses discrepantes pode prejudicar a

[4] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

revocação, especialmente se o relatório de consulta for um desses extremos, onde, independentemente do k usado, seria difícil recuperar uma duplicata usando a abordagem do vizinho mais próximo. Portanto, esses *outliers* devem ser verificados em experimentos futuros para revisar quais casos são discrepantes verdadeiros, pois podem ser uma duplicata rotulada incorreta ou uma descrição insuficiente do relatório de defeito, transformando-os em *outliers*, como afirmam Sadat et al. (2017).

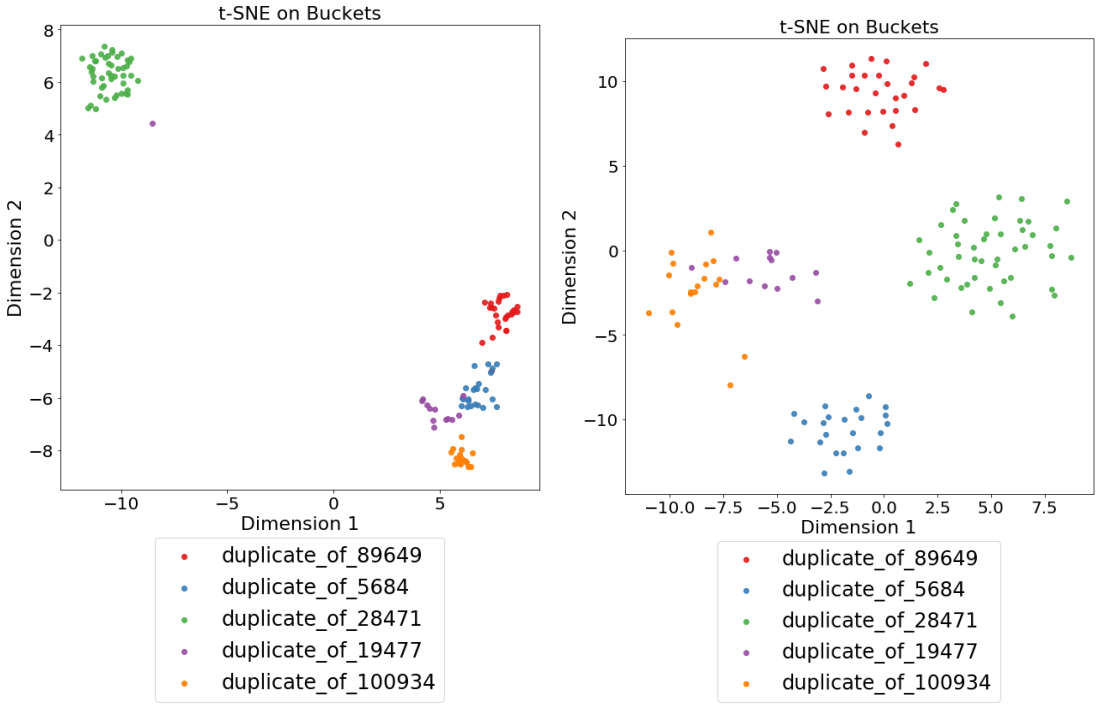
Como uma maneira automática de avaliar todos os *clusters* para cada conjunto de dados, na Tabela 19, podemos ver que mesmo SiameseQAT ou SiameseTA produzem grupos mais separáveis e bem definidos no espaço latente de acordo com a pontuação *Silhouette*. O resultado confirma que o uso de aprendizado semântico baseado em contexto em conjunto com *Quintet Loss* pode produzir *clusters* duplicados melhores, impactando em tarefas de detecção das duplicatas, como classificação e recuperação.

5.7 Atenção Textual Semântica

Como alternativa para entender como a BERT está tentando reconhecer recursos de texto e usar o mecanismo de atenção para detectar relatórios duplicados, selecionamos dois exemplos de frases de relatórios duplicados:

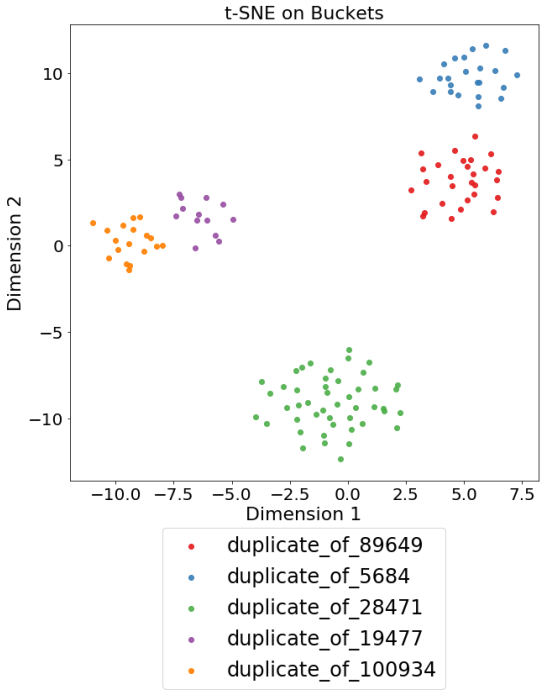
1. Frequent workspace crashes;
2. Out of memory error with Mylar?;

Às duas frases descrevem um problema incerto de "error", frequentemente causando falhas no aplicativo. Portanto, esperamos que o modelo BERT se concentre nas palavras "error" e "crashes" para determinar às duas frases como um contexto semelhante. Para nos permitir visualizar esse aprendizado, usamos a ferramenta BertViz criada por Vig (2019a), que relaciona as palavra entre si para cada camada e cabeçalho presente no modelo BERT. O gráfico da Figura 33 mostra uma relação de palavras aprendidas pelo modelo onde linhas mais nítidas aparecem entre duas palavras significam mais atenção. A Figura 33 possui duas frases dispostas de lado, relacionando cada palavra da camada #10 na cabeça #10 do BERT entre elas.



(a) DMS

(b) SiameseTAT



(c) SiameseQAT

Figura 32 – Espaço de duplicatas do Open Office para 5 grupos de duplicatas.

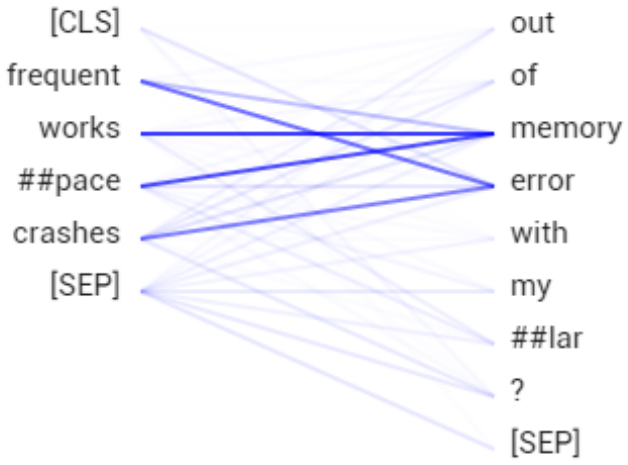


Figura 33 – Visualizando a camada #10, cabeça #10 do BERT pré-treinado para títulos de dois relatórios duplicados, usando a ferramenta de visualização BertViz criada por Vig (2019a).

Na Figura 33 podemos ver que o modelo dá atenção exatamente às palavras "memory, workspace, error" e "crashes", correlacionando também o "error" com "frequent" e "crashes", obtendo uma sentido global para o problema, ignorando todas as outras palavras auxiliares. Esse conhecimento é útil para auxiliar a discriminação entre às duas frases, para combiná-las como duplicatas.

5.8 Considerações Finais

Foram apresentados os experimentos para avaliar o SiameseQAT na tarefa detecção de duplicatas, tanto para classificação de pares duplicados, quanto para recuperação de relatórios duplicados. Nesses experimentos, também foram coletadas evidências e análises qualitativas sobre o efeito do uso da *Quintet Loss* e o mecanismo de atenção na tarefa de detecção, além das limitações da pesquisa. Os resultados mostram um cenário positivo para a tarefa de detecção das duplicatas, confirmando as hipóteses iniciais desta pesquisa. Em resumo, SiameseQAT apresenta resultados superiores aos trabalhos anteriores, na recuperação de duplicatas, obtendo revocação média de 41%-70% para Eclipse, 46%-73% para NetBeans e 43%-71% para Open Office. Nas tarefas de classificação, SiameseQAT apresentou acurácia de 96%, 95% e 94%, com AUCROC de

99% nas três bases, para Eclipse, NetBeans, e Open Office, respectivamente.

6 Conclusões

Neste trabalho apresentamos uma abordagem para detectar relatórios de defeitos duplicados por meio de duas tarefas, classificação e recuperação. Desse modo, analisando os trabalhos relacionados, constatamos que há uma tendência pelo uso de métodos baseados em aprendizagem profunda para detecção das duplicatas, por se tratar de uma abordagem que elimina engenharia manual para extrair significado dos atributos, além considerar relações semânticas entre o conteúdo dos relatórios, extraindo contexto e dissimilaridade vetorial entre os defeitos.

A partir da literatura introduzimos uma nova estrutura de detecção das duplicatas usando aprendizagem semântica baseada em contexto e a função de perda personalizada, Quintet Loss. As propostas levaram consistentemente a melhoras na precisão da recuperação, classificação e a qualidade dos grupos duplicados em três conjuntos de dados públicos, Eclipse, NetBeans e Open Office para mais de 500 mil relatórios de relatórios de defeitos. O modelo SiameseQAT é a primeira tentativa de apresentar uma função de perda customizada especificamente para aprendizado profundo e detecção de relatórios de defeitos duplicados, enquanto cobre a primeira tentativa de mesclar o contexto semântico baseado no BERT e os recursos de tópicos para detectar duplicatas. Os resultados de recuperação relatados foram superiores aos apresentados em trabalhos relacionados, com ganhos entre 12% e 25% na taxa de revocação, enquanto a acurácia e o AUROC superaram o melhor baseline, com 95% e 99% em média. Por fim, os resultados reportados mostram que SiameseQAT produz melhores grupos de duplicatas, impactando as tarefas de classificação e recuperação. No entanto, existem muitos trabalhos futuros que acreditamos melhorar ainda mais esse trabalho.

6.1 Limitações e Trabalhos Futuros da Pesquisa

Apesar dos resultados mostrarem uma superioridade do SiameseQAT em comparação aos baselines, nessa pesquisa não avaliamos a capacidade do modelo

de generalizar *embeddings* independente do projeto de software. Caso um novo projeto apareça com relatórios de jargões e vocabulários diferentes do treinado, não sabemos qual o impacto dos mesmos na qualidade da detecção das duplicatas. Na literatura, existem muitas bases que podem auxiliar essa experimentação como Firefox, Thunderbird, Cassandra, entre outras, disponíveis para uso[1]. Uma possível direção futura para atacar este problema seria o desenvolvimento de métodos de detecção de duplicatas que sejam treinados utilizando diversas bases de dados ao mesmo tempo, objetivando aprender representações que sejam válidas para diferentes repositórios e possivelmente utilizáveis em novos repositórios sem a necessidade de um retreinamento do zero.

Outro fato não abordado em nossa pesquisa são os *outliers*. Se existem os grupos de duplicatas que possuem esses relatórios irregulares, como afirmam Sadat et al. (2017), remover esses exemplos pode levar a uma melhoria no aprendizado, visto que eles forçam um ajuste do modelo para aprender padrões irregulares sobre o treino. Nesse sentido, não removemos esses relatórios da base por não sermos capazes de julgar quais são de fato irregulares, e principalmente porque essa análise necessita de um especialista de domínio ou um método de classificação automática para essa tarefa. Além disso, existem relatórios com ruídos no conjunto de dados, com o ID de referência duplicata referenciando um registro inexistente no conjunto de dados. Esse cenário precisa ser investigado, para entender se foi um erro de coleta pela Lazar et al. (2014), ou uma remoção intencional do relatório no sistema SRP. Pois, um determinado relatório pode ser referenciado por outro como duplicado. Dessa maneira, o mesmo relatório removido pode ser apagado do sistema permanecendo suas referências nas duplicatas.

Outra limitação em nossa abordagem é o processamento do tamanho das sequências textuais. A partir do atributo *description* dos relatórios, temos conjuntos de dados como NetBeans, com descrições de tamanho próximo a 800 palavras, devido a *stack trace*, etapas para reprodução do problema e código-fonte colados como evidência no conteúdo. Em nossa abordagem, nós processamos apenas as 100 primeiras palavras, apesar de termos feitos

[1] https://github.com/logpai/bugrepo

experimentos preliminares com 200. Todavia, por questões de recursos disponíveis, não conseguimos aumentar esse tamanho. Portanto, processar toda a informação textual de relatórios de defeitos é um desafio a ser resolvido.

Em relação a modelos neurais, a BERT permite treinar em blocos de no máximo 512, como recomenda Devlin et al. (2018). No entanto, em nossos experimentos estamos limitados a blocos de 64 no máximo, devido a esparsidade das outras características de um relatório de defeito, como título, descrição, tópicos e atributos categóricos, processados ao mesmo tempo em memória no SiameseQAT.

Em matéria da função *Quintet Loss*, apesar de agrupar melhor as duplicatas no espaço latente de características, identificamos alguns centroides idênticos a exemplos negativos no bloco de treinamento, devido ao fato de nenhuma outra duplicata desse mesmo exemplo negativo aparecer nesse lote de treinamento. Portanto, o centroide desse exemplo é sempre ele mesmo, uma vez que nenhuma réplica do mesmo grupo exista no bloco de treino. Nesse caso, é necessária outra estratégia de seleção dos exemplos negativos, que garanta a construção de centroides mais representativos no bloco, e consequentemente, impactar no aprendizado para que seja avaliado o desempenho desse bloco no treinamento do SiameseQAT.

Em relação às entradas do SiameseQAT, o título, descrição, tópicos e atributos categóricos, existem também os campos *stack trace*, *logs* e trechos de código-fonte em relatórios de defeitos, que estão presente no SRP, mas nas bases coletadas por Lazar et al. (2014) não contemplam esses campos separados do título e descrição, assim como foi feito por Wang et al. (2008). Portanto, não processamos esse tipo de informação separada em nossa arquitetura, o que acreditamos levar a melhorias na detecção das duplicatas se forem coletadas para treinar separadamente na arquitetura.

Referências

AGGARWAL, K. et al. Detecting duplicate bug reports with software engineering domain knowledge. In: IEEE. *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. [S.l.], 2015. p. 211–220. 72, 74, 75

AGGARWAL, K. et al. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 29, n. 3, p. e1821, 2017. 22, 23

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 46

BISHOP, C. M. *Pattern recognition and machine learning*, springer Cambridge, UK, v. 4, 2006. 39, 41, 43

BLEI, D. M.; NG, A. Y.; JORDAN, M. I. Latent dirichlet allocation. *Journal of machine Learning research*, v. 3, n. Jan, p. 993–1022, 2003. 57

BOMMASANI, R.; DAVIS, K.; CARDIE, C. Bert wears gloves: Distilling static embeddings from pretrained contextual representations. 2019. 62

BROMLEY, J. et al. Signature verification using a"siamese"time delay neural network. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1994. p. 737–744. 50

BROWNLEE, J. *Why One-Hot Encode Data in Machine Learning*. [S.l.]: Dostopno na: https://machinelearningmastery. com/why-one-hot-encode-data-in-machine-learning/, ogled, 2018. 63

BRUCH, S. et al. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. [S.l.: s.n.], 2019. p. 75–78. 52

BUDHIRAJA, A. et al. Dwen: deep word embedding network for duplicate bug report detection in software repositories. In: ACM. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. [S.l.], 2018. p. 193–194. 28, 37, 39, 52, 54, 62, 64, 66, 73, 74, 75, 99, 100, 103, 104

BUDHIRAJA, A. et al. Towards word embeddings for improved duplicate bug report retrieval in software repositories. In: *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*. [S.l.: s.n.], 2018. p. 167–170. 99, 100

CHECHIK, G. et al. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, v. 11, n. Mar, p. 1109–1135, 2010. 93

CHOLLET, F. et al. *Keras*. 2015. Https://keras.io. 97

CHOPRA, S.; HADSELL, R.; LECUN, Y. Learning a similarity metric discriminatively, with application to face verification. In: IEEE. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. [S.l.], 2005. v. 1, p. 539–546. 50

CHOROWSKI, J. K. et al. Attention-based models for speech recognition. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 577–585. 46

DASZYKOWSKI, M.; WALCZAK, B.; MASSART, D. Representative subset selection. *Analytica chimica acta*, Elsevier, v. 468, n. 1, p. 91–103, 2002. 72

DATA SCIENCE ACADEMY. Deep learning book. In: _____. *Deep Learning Book*. 2019. v. 1. Disponível em: <http://deeplearningbook.com.br/arquitetura-de-redes-neurais-long-short-term-memory/>. Acesso em: 22 jun. 2020. 41

DAVID, H. A.; GUNNINK, J. L. The paired t test under artificial pairing. *The American Statistician*, Taylor & Francis, v. 51, n. 1, p. 9–12, 1997. 67

DESHMUKH, J. et al. Towards accurate duplicate bug retrieval using deep learning techniques. In: IEEE. *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. [S.l.], 2017. p. 115–124. 28, 30, 37, 40, 41, 44, 46, 50, 53, 54, 62, 64, 65, 73, 74, 75, 89, 91, 92, 93, 100, 103

DEVASSY, B. M.; GEORGE, S. Dimensionality reduction and visualisation of hyperspectral ink data using t-sne. *Forensic Science International*, Elsevier, p. 110194, 2020. 67

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 28, 46, 47, 48, 55, 104, 120

FAYYAZ, M. et al. Stfcn: Spatio-temporal fcn for semantic video segmentation. 08 2016. 41, 42

GE, W. Deep metric learning with hierarchical triplet loss. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 269–285. 30, 93

GONZALEZ, R. C.; WOODS, R. E. Digital image processing (preview). Prentice Hall, 2002. 44

GOODFELLOW, I. et al. *Deep learning*. [S.l.]: MIT press Cambridge, 2016. v. 1. 37, 39, 84, 87

IMANI, A. et al. Deep neural networks for query expansion using word embeddings. In: SPRINGER. *European Conference on Information Retrieval*. [S.l.], 2019. p. 203–210. 49

JADERBERG, M. et al. Spatial transformer networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 2017–2025. 44, 45

JAIN, S.; WALLACE, B. C. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019. 46, 47, 85, 104

JALBERT, N.; WEIMER, W. Automated duplicate detection for bug tracking systems. In: IEEE. *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. [S.l.], 2008. p. 52–61. 71, 72, 75

JING, L.; TIAN, Y. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 2020. 37

JOULIN, A. et al. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016. 85

KIM, Y. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 44, 45

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2014. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. Disponível em: <http://arxiv.org/abs/1412.6980>. 102

KLEIN, N.; CORLEY, C. S.; KRAFT, N. A. New features for duplicate bug detection. In: ACM. *Proceedings of the 11th Working Conference on Mining Software Repositories*. [S.l.], 2014. p. 324–327. 28

LAN, W.; XU, W. Neural network models for paraphrase identification, semantic textual similarity, natural language inference, and question answering. *arXiv preprint arXiv:1806.04330*, 2018. 46, 49

LAZAR, A.; RITCHEY, S.; SHARIF, B. Generating duplicate bug datasets. In: ACM. *Proceedings of the 11th working conference on mining software repositories*. [S.l.], 2014. p. 392–395. 12, 71, 72, 74, 79, 80, 83, 84, 98, 119, 120

LEE, S.-R. et al. Applying deep learning based automatic bug triager to industrial projects. In: ACM. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. [S.l.], 2017. p. 926–931. 33

LI, X. et al. Unsupervised deep bug report summarization. In: ACM. *Proceedings of the 26th Conference on Program Comprehension*. [S.l.], 2018. p. 144–155. 25, 33

LINDERMAN, G. C.; STEINERBERGER, S. Clustering with t-sne, provably. *SIAM Journal on Mathematics of Data Science*, SIAM, v. 1, n. 2, p. 313–332, 2019. 67

LIU, W. et al. Large-margin softmax loss for convolutional neural networks. In: *ICML*. [S.l.: s.n.], 2016. v. 2, n. 3, p. 7. 52

LUKINS, S. K.; KRAFT, N. A.; ETZKORN, L. H. Bug localization using latent dirichlet allocation. *Information and Software Technology*, Elsevier, v. 52, n. 9, p. 972–990, 2010. 59

LUONG, M.-T.; PHAM, H.; MANNING, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015. 46

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. [S.l.: s.n.], 2013. v. 30, n. 1, p. 3. 96

MAATEN, L. v. d.; HINTON, G. Visualizing data using t-sne. *Journal of machine learning research*, v. 9, n. Nov, p. 2579–2605, 2008. 67

MANI, S.; SANKARAN, A.; ARALIKATTE, R. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In: *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*. [S.l.: s.n.], 2019. p. 171–179. 42

MANSANO, A. F. et al. Aprendizado de máquina multivisão aplicado à análise de correferência em um sistema de aprendizado sem fim. Universidade Federal de São Carlos, 2018. 60, 61

MASSON, E.; WANG, Y.-J. Introduction to computation and learning in artificial neural networks. *European Journal of Operational Research*, Elsevier, v. 47, n. 1, p. 1–28, 1990. 37

MIKOLOV, T. et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. 60

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: BURGES, C. J. C. et al. (Ed.). *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013. p. 3111–3119. Disponível em: <http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>. 60

MULLER, B.; SAGOT, B.; SEDDAH, D. Enhancing bert for lexical normalization. In: *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*. [S.l.: s.n.], 2019. p. 297–306. 55

NGUYEN, A. T. et al. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: ACM. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2012. p. 70–79. 28, 29, 59, 71, 75, 89

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA:, 2015. v. 25. 51

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1310–1318. 41

PENNINGTON, J.; SOCHER, R.; MANNING, C. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. [S.l.: s.n.], 2014. p. 1532–1543. 62, 85

PETERS, M. E. et al. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 48

QIN, H.; SUN, X. Classifying bug reports into bugs and non-bugs using lstm. In: . [S.l.: s.n.], 2018. p. 1–4. ISBN 978-1-4503-6590-1. 41

RANASINGHE, T.; ORASAN, C.; MITKOV, R. Semantic textual similarity with siamese neural networks. In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*. [S.l.: s.n.], 2019. p. 1004–1011. 49

REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019. 49, 51

ROBERT, C. *Machine learning, a probabilistic perspective*. [S.l.]: Taylor & Francis, 2014. 52

ROCHA, H. et al. An empirical study on recommendations of similar bugs. In: IEEE. *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*. [S.l.], 2016. v. 1, p. 46–56. 28

RUNESON, P.; ALEXANDERSSON, M.; NYHOLM, O. Detection of duplicate defect reports using natural language processing. In: IEEE COMPUTER SOCIETY. *Proceedings of the 29th international conference on Software Engineering*. [S.l.], 2007. p. 499–510. 34, 70, 75

SABOUR, S. *Dynamic routing between capsules, source code (2017)*. 2019. 44, 45

SADAT, M.; BENER, A. B.; MIRANSKYY, A. V. Rediscovery datasets: Connecting duplicate reports. In: IEEE PRESS. *Proceedings of the 14th International Conference on Mining Software Repositories*. [S.l.], 2017. p. 527–530. 24, 114, 119

SALEHINEJAD, H. et al. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017. 42

SANTOS, C. dos; GATTI, M. Deep convolutional neural networks for sentiment analysis of short texts. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. [S.l.: s.n.], 2014. p. 69–78. 43

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. Disponível em: <http://arxiv.org/abs/1503.03832>. 30, 93

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 815–823. 54, 91, 93

SCHUSTER, M.; PALIWAL, K. K. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, Ieee, v. 45, n. 11, p. 2673–2681, 1997. 42

SHARMA, L. et al. *Natural Language Understanding with the Quora Question Pairs Dataset*. 2019. 80, 81

SINGH, J. et al. Bert is not an interlingua and the bias of tokenization. In: *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*. [S.l.: s.n.], 2019. p. 47–55. 48

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information processing & management*, Elsevier, v. 45, n. 4, p. 427–437, 2009. 65

STAVRIANOU, A.; ANDRITSOS, P.; NICOLOYANNIS, N. Overview and semantic issues of text mining. *ACM Sigmod Record*, ACM, v. 36, n. 3, p. 23–34, 2007. 55

STEHMAN, S. V. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, Elsevier, v. 62, n. 1, p. 77–89, 1997. 65

SULEA, O.-M. et al. Exploring the use of text classification in the legal domain. *arXiv preprint arXiv:1710.09306*, 2017. 65

SUN, C. et al. Towards more accurate retrieval of duplicate bug reports. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2011. p. 253–262. 28, 33, 71, 75, 82, 104

SUN, C. et al. A discriminative model approach for accurate duplicate bug report retrieval. In: ACM. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. [S.l.], 2010. p. 45–54. 70, 75

Terdchanakul, P. et al. Bug or not? bug report classification using n-gram idf. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.: s.n.], 2017. p. 534–538. 39

TODI, K. K.; MISHRA, P.; SHARMA, D. M. Building a kannada pos tagger using machine learning and neural network models. *arXiv preprint arXiv:1808.03175*, 2018. 42, 43

TRAN, N. K.; NIEDEREÉE, C. Multihop attention networks for question answer matching. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. [S.l.: s.n.], 2018. p. 325–334. 46

UDDIN, J. et al. A survey on bug prioritization. *Artificial Intelligence Review*, Springer, v. 47, n. 2, p. 145–180, 2017. 22, 34, 36

VASWANI, A. et al. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008. 46

VEEN, F. V. *The Neural Network Zoo*. 2016. <http://www.asimovinstitute.org/neural-network-zoo/>. Acessado em 10 de novembro de 2018. 37

VIG, J. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*, 2019. Disponível em: <https://arxiv.org/abs/1906.05714>. 11, 114, 116

VIG, J. Openai gpt-2: Understanding language generation through visualization. *Towards Data Science, via Medium, March*, v. 5, 2019. 48

VINYALS, O. et al. Show and tell: A neural image caption generator. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 3156–3164. 37

VIRTANEN, A. et al. Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*, 2019. 48, 57

WALPOLE, R. E. et al. *Probability and statistics for engineers and scientists*. [S.l.]: Macmillan New York, 1993. v. 5. 67

WANG, X. et al. An approach to detecting duplicate bug reports using natural language and execution information. In: ACM. *Proceedings of the 30th international conference on Software engineering*. [S.l.], 2008. p. 461–470. 71, 75, 120

WANG, Z.; MI, H.; ITTYCHERIAH, A. Semi-supervised clustering for short text via deep representation learning. *arXiv preprint arXiv:1602.06797*, 2016. 51

WU, C.-Y. et al. Sampling matters in deep embedding learning. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. p. 2840–2848. 30, 93, 94

XIA, L. et al. A survey of topic models in text classification. In: IEEE. *2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD)*. [S.l.], 2019. p. 244–250. 57

XUAN, J. et al. Automatic bug triage using semi-supervised text classification. *arXiv preprint arXiv:1704.04769*, 2017. 28

YIN, W. et al. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, MIT Press, v. 4, p. 259–272, 2016. 46

YOUNG, T. et al. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, IEEE, v. 13, n. 3, p. 55–75, 2018. 46, 49

YU, F.; KOLTUN, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 44, 45

Zafar, S.; Malik, M. Z.; Walia, G. S. Towards standardizing and improving classification of bug-fix commits. In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2019. p. 1–6. 29, 49

ZHANG, T. et al. A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, Oxford University Press, v. 59, n. 5, p. 741–773, 2016. 22, 23, 27, 33, 34, 36

ZHENG, A.; CASARI, A. Feature engineering for machine learning: Principles and techniques for data scientists. In: *Principles and Techniques for Data Scientists*. [S.l.: s.n.], 2018. p. 218. 63