UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RYAN CAMARGO LUNA

AGRUPAMENTO DE *BUGS* DE SOFTWARE A PARTIR DO MODELO DE LINGUAGEM PRÉ-TREINADO BERT E MÉTODOS DE *CLUSTERING*

PATO BRANCO
2022

RYAN CAMARGO LUNA

AGRUPAMENTO DE *BUGS* DE SOFTWARE A PARTIR DO MODELO DE LINGUAGEM PRÉ-TREINADO BERT E MÉTODOS DE *CLUSTERING*

GROUPING OF SOFTWARE BUGS BASED ON THE PRE-TRAINED BERT LANGUAGE MODEL AND CLUSTERING METHODS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientadora: Dra. Eliane Maria De Bortoli Fávero.

PATO BRANCO
2022

RYAN CAMARGO LUNA

AGRUPAMENTO DE *BUGS* DE SOFTWARE A PARTIR DO MODELO DE LINGUAGEM PRÉ-TREINADO BERT E MÉTODOS DE *CLUSTERING*

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de Aprovação: 21 de junho de 2022.

Dra. Eliane Maria de Bortoli Fávero
Universidade Tecnológica Federal do Paraná

Dr. Dalcimar Casanova
Universidade Tecnológica Federal do Paraná

Dr. Robison Cris Brito
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2022

Dedico este trabalho a Deus, minha família, e a minha orientadora.

AGRADECIMENTOS

Em primeiro lugar, quero agradecer à minha orientadora, Prof.ª Doutora Eliane Maria de Bortoli Fávero, pela sua disponibilidade em me mostrar os caminhos a serem seguidos e pela confiança depositada.

O presente trabalho não poderia ser finalizado sem a ajuda de diversas pessoas às quais presto meus agradecimentos. A minha família, pelo carinho, incentivo e total apoio em todos os momentos da minha vida.

A todos os professores e colegas do curso, que ajudaram de forma direta e indireta na realização deste trabalho.

A todos os demais que de alguma forma contribuíram para meu crescimento pessoal e profissional.

Deus não escolhe os capacitados, capacita os escolhidos. Fazer ou não fazer algo só depende de nossa vontade e perseverança. - Albert Einsten

RESUMO

A necessidade de manutenção de software após sua implantação, ou mesmo a dificuldade dos usuários em usar um aplicativo de software é uma realidade até os dias atuais. Um problema de usuário ou *bug* de software, trata-se de um relato de um usuário de que determinado procedimento do sistema não funciona como o esperado. Algumas vezes o fato relatado se refere a um problema a ser resolvido, seja de implementação, de configuração do software, de falta de conhecimento do usuário, o qual já ocorreu e teve sua solução documentada. Esse fato facilita e agiliza o atendimento por parte do pessoal do suporte, não sendo necessário enviar o problema para outros departamentos da empresa ou investir muito tempo na sua solução. Sendo assim, o presente trabalho objetiva classificar esses problemas de software com base em textos de solicitações de usuários ao departamento de suporte, a fim de retornar possíveis soluções de forma rápida. Para isso foi criada uma base de dados de *bugs* de software na língua portuguesa, sobre a qual foram aplicados métodos de Processamento de Linguagem Natural (PLN) para realizar o pré-processamento e a representação textual dos textos de *bugs*. A representação textual ocorreu pela aplicação do modelo pré-treinado contextualizado *Bidirectional Encoder Representations from Transformers* (BERT) em sua versão BERT_base. A representação dos textos foi usada na aplicação do método de agrupamento *Density-Based Clustering Based on Connected Regions with High Density* (DBSCAN), a fim de classificá-los. Após realizar testes com diferentes valores de hiperparâmetros e métodos de redução de dimensionalidade, os resultados revelaram que não existe uma estrutura subjacente a partir da representação de dados aplicada. Uma hipótese para o resultado alcançado é a de que o modelo pré-treinado aplicado não é capaz de representar adequadamente o contexto dos textos de *bugs* em português. Trabalhos futuros são propostos buscando encontrar métodos mais eficazes para o objetivo proposto.

Palavras-chave: PLN; Problema; *Bug*; Usuário; Software.

ABSTRACT

The need of software maintenance after its implementation, even the user's difficulties in using a software application is a reality to this day. A user's problem or software bug, it is a user's relate that a particular procedure of the system didn't work as expected. Some times the related fact concerns to a problem to be solved (e.g. of implementation, configuration of software, lack of users expertise), which have been occurred and had its solution documented. This fact facilitates and speeds up the service from the support team, being not necessary to send the problem to others enterprise departments or spend a lot of time in its solution. Therefore, the present work aims to classify these software problems based in text of users requirements to the support department, in order to find potential solutions rapidly. For this it was created a data base of software bugs in Portuguese, on which the PLN methods were applied to perform the pre-processing and the textual representation of texts of bugs. The textual representation occurred by applying the contextualized pre-trained model BERT in its BERT_base version. The representation of the texts was used in the application of the clustering method DBSCAN, in order to classify them. After performing tests with different hyperparameter values and dimensionality reduction methods, the results revealed that there is no underlying structure from the applied data representation. One hypothesis for the result achieved is that the pre-trained model applied is not able to adequately represent the context of *bugs* texts in Portuguese. Future works are proposed seeking to find more effective methods for the proposed objective.

Keywords: PLN; Problem; Bug; User; Software.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

BERT	*Bidirectional Encoder Representations from Transformers*
BOW	*Bag-of-word*
DBSCAN	*Density-Based Clustering Based on Connected Regions with High Density*
IA	Inteligência artificial
ITIL	*Information Technology Service Library*
NLTK	*Natural Language Toolkit*
ODC	Classificação Ortogonal de Defeitos
PCA	*Principal component analysis*
PLN	Processamento de Linguagem Natural
POS	*Part-Of-Speech*
TI	Tecnologia da Informação
TSN-e	*T-Distributed Stochastic Neighbor Embedding*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a contextualização que envolve o problema de pesquisa proposto para a realização deste Trabalho de Conclusão de Curso, o qual se refere à aplicação de métodos de PLN, objetivando propor uma metodologia semiautomática para auxiliar o departamento de suporte de empresas de software, no atendimento às solicitações dos usuários de seus softwares na resolução de problemas (*bugs*).

1.1 Considerações Iniciais

A palavra *bug* é utilizada para identificar problemas inesperados ocorridos em hardware ou software durante a sua utilização. De acordo com Leitão (2016), um *bug* é um estado compu- tacional não previsto que pode levar à uma falha e pode causar, tanto erros físicos (no hardware), quanto erros criados a partir de defeitos de programação.

Um problema comum na área de Tecnologia da Informação (TI), especialmente em softwares, é produzir sistemas que possam ser confiáveis para seus usuários e que tenham a proteção necessária contra falhas, as quais podem ser localizadas em qualquer etapa do seu desenvolvimento, porém, deve-se evitar que essas falhas cheguem até o usuário final, o que im- pacta diretamente em seu nível de qualidade. Caso isso ocorra pode se tratar de um problema, dúvida ou falta da configuração correta, fazendo com que o usuário recorra ao suporte técnico da empresa desenvolvedora. No contexto dessa proposta, um *bug* se refere e será denominado problema de software, sendo caracterizado por um evento que altera o estado interno de um sistema de software para um estado em que o serviço não corresponde ao esperado ou desejado, conforme o definido em alguma especificação (BARBOSA, 2012).

Normalmente, o usuário requer respostas rápidas e precisas, e que resolvam de forma efetiva o seu problema. Por conta disso, muitas vezes, o suporte sofre uma alta demanda por respostas às solicitações, tanto decorrentes da falta de conhecimento do cliente, quanto por falhas no software ou ainda necessidades de melhorias em seus processos. Uma das ferramentas de auxílio ao departamento de suporte são os chamados sistemas de *help-desk*, os quais possuem a função de contribuir para aprimorar o trabalho da equipe de suporte das empresas, auxiliando na coordenação e solução de problemas/dificuldades trazidas pelos usuários, com o objetivo de atender e resolver estes incidentes com eficácia e eficiência. Além disso, sistemas de *help-desk* (em português "balcão de ajuda") atuam como um elo entre a empresa e o cliente. Esse tipo de sistema exerce papel especialista, empregando os registros cadastrados como base para apoio às decisões em todos os níveis de suporte, facilitando os atendimentos e fazendo-os com qualidade (LINKE, 2015).

Um sistema de *help-desk* automatizado ou semiautomatizado, permitiria agrupar uma lista de *bugs* atendidos no passado, e que fossem mais similares à solicitação atual de um dado usuário. Isso seria importante, pois tornaria o atendimento mais ágil, e consequentemente au-

mentaria a sua qualidade, além de diminuir casos de *bugs* que não procedem. Outro benefício desse tipo de sistema, seria a otimização do tempo por parte dos desenvolvedores, pois com o modelo de aprendizado, haverá uma espécie de filtro, aliado ao trabalho do suporte, para que um problema de usuário só chegue ao desenvolvimento, caso realmente haja necessidade.

A mineração de textos é usada para realizar uma análise inteligente do texto escrito com o objetivo de automatizar ou auxiliar outros processos (ex. agrupamento automático de textos, resumo automático) (GHAREHCHOPOGH; KHALIFELU, 2011). Sendo assim, a mineração de texto é o processo de análise de texto para extrair informações que são úteis para fins espe- cíficos. O agrupamento de textos gera interesse tanto em pesquisadores da área de Aprendizado de Máquina e Recuperação de Informação, quanto em desenvolvedores que precisam trabalhar com grandes quantidades de documentos no formato textual, muitas vezes disponíveis em di- versas bases de dados, contendo documentos de diferentes tipos e com variadas quantidades de informação.

Gomes (2013) pontua a relevância da aplicação de métodos de mineração de textos para a extração de conhecimento em bases de dados textuais, uma vez que a mineração de dados é comumente aplicada em dados que possuem certo nível de estruturação e contemplam apenas uma parte limitada de dados que as organizações possuem, ou seja, dados estruturados. Desta forma, a mineração de textos busca extrair representações de textos em linguagem natural, e para isso requer a aplicação de métodos de PLN, uma subárea da Inteligência artificial (IA) que permite aos computadores processar a linguagem natural e entender o significado da linguagem humana (DUMAS; LALANNE; OVIATT, 2009). O PLN é uma área da computação que tem como objetivo extrair representações e significados mais completos de textos livres escritos em linguagem natural (BEHZADI, 2015).

Alguns pesquisadores já desenvolveram trabalhos similares buscando auxiliar o traba- lho em sistemas de *help-desk*, tais como: Sousa *et al.* (2016) que construíram um agrupamento automático de severidade de *bugs* para sistemas *open source*, a base de dados já era rotulada, ou seja, tratava-se de um modelo supervisionado, o qual baseou-se em três algoritmos (Naive Bayes, Max Ent e Winnow) para aferir se a severidade normal estava correta para os *bugs* assim categorizados (urgência do *bug*). Pereira e Christiansen (2020) fizeram a classificação dos rela- tórios de *bugs*, efetuando um estudo comparativo de robustez das três técnicas de aprendizado de máquina (regressão logística, Naive Bayes e AdaBoost), utilizando aprendizado de máquina supervisionado.

Bandeira *et al.* (2019) trabalhou com a classificação automática da prioridade de defeitos de software utilizando uma base de dados do Jira (rotulada), com abordagens de seleção de atributos utilizando a USES+ (versão melhorada do USES, onde é calculado o score de afinidade entre categoria e palavra, que é calculado de acordo com a existência ou não da palavra nos registros previamente agrupados. No USES+, o score é simplesmente a relação entre o número de atributos de uma categoria com uma determinada palavra e a quantidade total de atributos com essa palavra), resultando em boa efetividade. Dessa forma, o agrupamento automático

permite a redução do custo de priorização dos defeitos, melhorando significativamente o tempo para as correções.

Observando os trabalhos similares já realizados, pode-se dizer que o diferencial do trabalho proposto é a geração de uma base de dados de *bugs* em português, visto que há uma carência de estudos nesta linguagem. Outro diferencial possível é a aplicação de uma forma de representação textual diferente para essa finalidade, bem como a aplicação de métodos de aprendizado de máquina não-supervisionados. O método de representação textual a ser utilizado é o BERT, um modelo de linguagem que não foi aplicado por nenhum outro trabalho similar. Com isso, o objetivo deste trabalho é propor uma metodologia semiautomática para auxiliar o departamento de suporte de empresas de software, no atendimento às solicitações dos usuá- rios de seus softwares. A proposta para essa solução se baseia no agrupamento de *bugs* com base em textos de problemas passados, todos na língua portuguesa, aplicando métodos de PLN associados ao aprendizado de máquina.

Com relação às formas de representação textual, buscou-se aplicar modelos de linguagem pré-treinados para a geração de *embeddings* para cada sentença representativa de um *bug*. Mais especificamente, é aplicado um modelo de *embedding* pré-treinado baseado fortemente em contexto, o BERT. O BERT é o modelo pré-treinado da Google para PLN, o qual se utiliza de aprendizado profundo para ajudar computadores e máquinas a entender a linguagem como os humanos. É um modelo de representação de linguagem projetado para pré-treinar represen- tações bidirecionais profundas de texto não rotulado (DEVLIN *et al.*, 2018). Assim, o BERT objetiva gerar uma representação no formato numérico (*embedding*) para cada palavra dentro do contexto de uma frase.

Desta forma, a partir de dados textuais de *bugs* fornecidos como entrada pelo usuário e convertidos para *embeddings* por meio do modelo pré-treinado BERT, pretende-se agrupar um subconjunto de problemas similares, fazendo uso de algoritmos de agrupamento. Assim, o pessoal do suporte poderá selecionar uma solução mais rapidamente para que seja retornada ao usuário. Ou ainda, o pessoal de desenvolvimento receberá uma demanda com possível solução já associada, agilizando ainda mais o seu trabalho. Isso seria possível, porque a proposta de sis- tema aqui apresentada objetiva realizar o agrupamento com base em analogia (WANG; YANG, 2011), ou seja, estaria retornando respostas possíveis, atribuídas às solicitações já resolvidas, e que ficaram armazenadas na base de dados da empresa. Um sistema como esse, poderá re- duzir e agilizar o trabalho do suporte sobre as altas demandas, além de facilitar o trabalho dos desenvolvedores sobre problemas já ocorridos.

1.2 Objetivos

A seguir serão apresentados o objetivo geral e específicos desse trabalho. O objetivo geral explica o resultado principal que se espera para o projeto, já os objetivos específicos, detalham os subprodutos gerados a partir do objetivo geral, desenvolvidos ao longo do projeto.

1.2.1 Objetivo Geral

- Propor uma metodologia semiautomática para agrupamento de *bugs* de software aplicando o modelo pré-treinado BERT associado a métodos de aprendizado de máquina.

1.2.2 Objetivos Específicos

- Gerar uma base de dados de textos de *bugs* de software em português;
- Gerar a representação textual da base de textos fazendo uso do modelo pré-treinado BERT;
- Aplicar métodos de agrupamento para gerar *clusters* significativos a partir de *bugs*;
- Analisar os resultados obtidos e apresentar as considerações devidas, objetivando a proposta de uma metodologia a ser aplicada em um sistema de apoio para o departamento de suporte, que apresente sugestões de respostas aos problemas dos usuários com base em sua representação textual.

1.3 Justificativa

Considerando as constantes mudanças e o crescimento contínuo para manter a utilidade, a qualidade dos serviços de softwares, e principalmente os clientes, Riveros, Campos e Peraz- zolli (2017) dizem que durante um problema ou incidente, o usuário busca por uma solução rápida para que o problema seja resolvido. Muitas vezes o usuário perde até horas interagindo com os setores da empresa até que uma solução seja encontrada. Além disso, muitos *bugs* não procedem, fazendo com que a equipe de desenvolvimento invista um tempo desnecessário na solução de algo que não seria de sua responsabilidade. Isso ocorre, muitas vezes, por falta de experiência do atendente do suporte, por falta de um histórico sistematizado, que retorne rapi- damente possíveis soluções e direcionamentos.

Sendo assim, objetivando agilizar o processo de atendimento e reduzir o esforço desne- cessário do setor de suporte, desenvolvedores, entre outros colaboradores envolvidos, a proposta de uma metodologia possível de ser implementada em um sistema de *help-desk*, que o torna se- miautomatizado, seria de suma importância.

O agrupamento de textos, quando realizada por um computador e utilizando os métodos de PLN, consiste em detectar a categoria a qual pertence determinado problema de software com base nas características obtidas dos textos e na probabilidade dessas características perten- cerem a determinadas classes pré-definidas (RUSSEL; NORVIG, 2013). Outra possibilidade é a geração de agrupamentos de textos similares quanto à sua semântica, o que retornaria um agru-

pamento automático, agrupando *bugs* similares, facilitando muito a realização das atividades do usuário.

1.4 Estrutura do trabalho

Este trabalho está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico e as tecnologias abordadas neste trabalho. No Capítulo 3 estão os materiais e o método para a realização do trabalho. No Capítulo 4 se trata dos resultados obtidos após a implementação do método proposto. No Capítulo 5 são apresentadas as conclusões obtidas após análise dos resultados, bem como sugestões de trabalhos futuros. E por fim estão as referências utilizadas no texto.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico necessário para o desenvolvimento deste estudo. As seções a seguir, apresentam os conceitos e métodos aplicados no desenvolvimento desse trabalho. Primeiramente, na seção 2.1 são apresentados o contexto e os conceitos acerca de *bugs* presentes nos softwares e a caracterização de sistemas *help-desk*. Na seção 2.2 são abordados os conceitos e aplicações PLN e as diversas aplicações dessa área. Na seção 2.3 são apresentados os conceitos relacionados com o pré-processamento de textos, conforme utilizados nesse trabalho. Na seção 2.6 relatamos o algoritmo que fora utilizado no trabalho. Já na seção 2.7 é comentado de dois métodos utilizados para avaliação do algoritmo DBSCAN. Na seção 2.5 traz os conceitos e métodos de aprendizado de máquina, com destaque para aqueles que foram aplicados neste estudo. E por fim a seção 2.8 traz um relato de sistemas já existentes e similares ao trabalho realizado.

2.1 Bugs de Software e Sistemas de *Help-desk*

Bug se refere a alguma falha que ocorrem ao executar algum software, e no hardware. Existem erros que prejudicam o funcionamento, trazendo variações de problemas ou inconsistências, fazendo com que ocasione travamentos, mau funcionamento ou, simplesmente, não funcione nada. Uma falha pode causar tanto erros físicos no hardware, quanto erros criados a partir de deslizes de programação (LEITÃO, 2016).

Segundo o IEEE (2021), os *bugs* são classificados em diferentes categorias, sendo: defeito, erro, falha e falta. As categorias são definidas desta forma e Barbosa (2012) define como:

- Defeito: É um evento que altera o estado interno de um sistema de software para um estado em que o serviço não corresponde ao esperado, ou desejado, conforme o definido em alguma especificação.
- Erro: É um estado interno de um sistema de software que possibilita a ocorrência de um defeito.
- Falha: É uma causa hipotética de erro. Falhas podem ser imperfeições ou irregularidades que ocorrem em módulos de hardware ou software).
- Falta: É a causa física ou algorítmica de um erro.

No contexto de *bugs* de softwares, os sistemas de *help-desk*, é um apoio de prestação de suporte remoto, que visa a resolução de problemas de informática, telefonia, software, existentes nos usuários. Este apoio pode ser tanto dentro de uma empresa (profissionais que cuidam da manutenção de equipamentos e instalações dentro da empresa), quanto externamente (prestação de serviços a usuários) conforme Filho (2017).

Segundo Filho (2017), o analista ou suporte de *help-desk* atende as solicitações dos cli- entes por telefone, e-mail ou até mesmo via acesso remoto ao ambiente do cliente. Foi criado em 1989 um instituto para aperfeiçoamento de técnicas para tal atendimento, hoje então, co- nhecido como Help Desk Institute, uma instituição profissional que tinha como missão servir o mercado, tendo o seu foco em inovar no suporte técnico (MUNS, 1993).

Help Desk Institute se tornou uma associação global de referência no desenvolvimento do segmento de atendimento e suporte a clientes internos e externos. Por meio de cursos/treina- mentos e certificações de centrais de suporte a cliente, auxiliando na evolução dos profissionais individualmente e das operações de serviços de suporte (OLIVEIRA, 2019).

Contudo, mesmo após décadas, o grande desafio de quem atua na área de suporte é se fazer entender e ser compreendido, transformando uma linguagem técnica em uma linguagem compreensível para o usuário (OLIVEIRA, 2019). Ainda para o Oliveira (2019), para a comu- nicação estar clara é preciso uma estrutura que contemple em seus processos:

- Conhecer o usuário;
- Saber as dificuldades que o usuário apresenta;
- Consultar as bases de conhecimento sobre os sistemas e equipamentos aos quais se presta o suporte, se necessário, para solucionar o problema do usuário;
- Responder aos questionamentos dos usuários evitando utilizar termos técnicos;
- Verificar se, nas dificuldades apresentadas pelo usuário, aparece código de erro;
- Fazer a transcrição da chamada/passos orientados para resolução do problema técnico;
- Construir uma base de conhecimento para o próprio operador e compartilhar com a equipe de suporte;
- Fazer testes com atualizações do produto;
- Juntar as informações das necessidades do usuário para fazer uma atualização do produto e/ou serviço.

No contexto do presente trabalho, a mineração de textos surge como uma potencial ferramenta de auxílio à tomada de decisões e gestão de recursos. Por exemplo, sabe-se que para todo atendimento ao usuário de software, são feitos registros em formato texto livre, dos *bugs* encontrados, bem como o registro da solução oferecida pelo suporte. A mineração desses registros textuais, pode viabilizar a antecipação da tomada de decisões, melhorando a gestão do fluxo de tempo no atendimento ao usuário. Furtado (2004) também descreve que a mineração de textos pode ser usada para formalizar e explorar o conhecimento tácito, proporcionando a descoberta de soluções, e de novos conhecimentos. As ferramentas de mineração de textos, têm o propósito de facilitar o processo de recuperação de informação, minimizando as dificuldades

enfrentadas e apresentando ao usuário algum tipo de conhecimento útil e novo, mesmo que tal conhecimento não seja a resposta direta, satisfazendo pelo menos as necessidades de novas informações.

2.2 Processamento de Linguagem Natural

O PLN é uma subárea da IA, que ao combinar grandes quantidades de dados com processamento rápido e interativo e algoritmos inteligentes, permite ao software aprender automa- ticamente com padrões ou informações nos dados. A IA é um campo de estudo amplo, que engloba muitas teorias, métodos e tecnologias. O PLN é responsável pela lapidação dos resul- tados, tornando-os mais naturais e mais humanos, fazendo combinações linguística computaci- onal, modelagem baseada em regras da linguagem natural, e também com modelos estatísticos, de aprendizado de máquina e de aprendizado profundo.

Desta forma, PLN estuda o desenvolvimento de programas de computadores capazes de analisar, reconhecer e/ou gerar textos em linguagens humanas, ou linguagens naturais (VIEIRA; LOPES, 2010). Esta área busca entender como o ser humano entende e usa determinada língua como, por exemplo, o inglês, a fim de projetar sistemas computacionais capazes de re- conhecimento e síntese da fala humana, análise léxico-morfológica e entre outros. De acordo com Chowdhury (2003), a área da computação que lida com informações fornecidas em língua natural, seja falada ou escrita, é denominada de PLN.

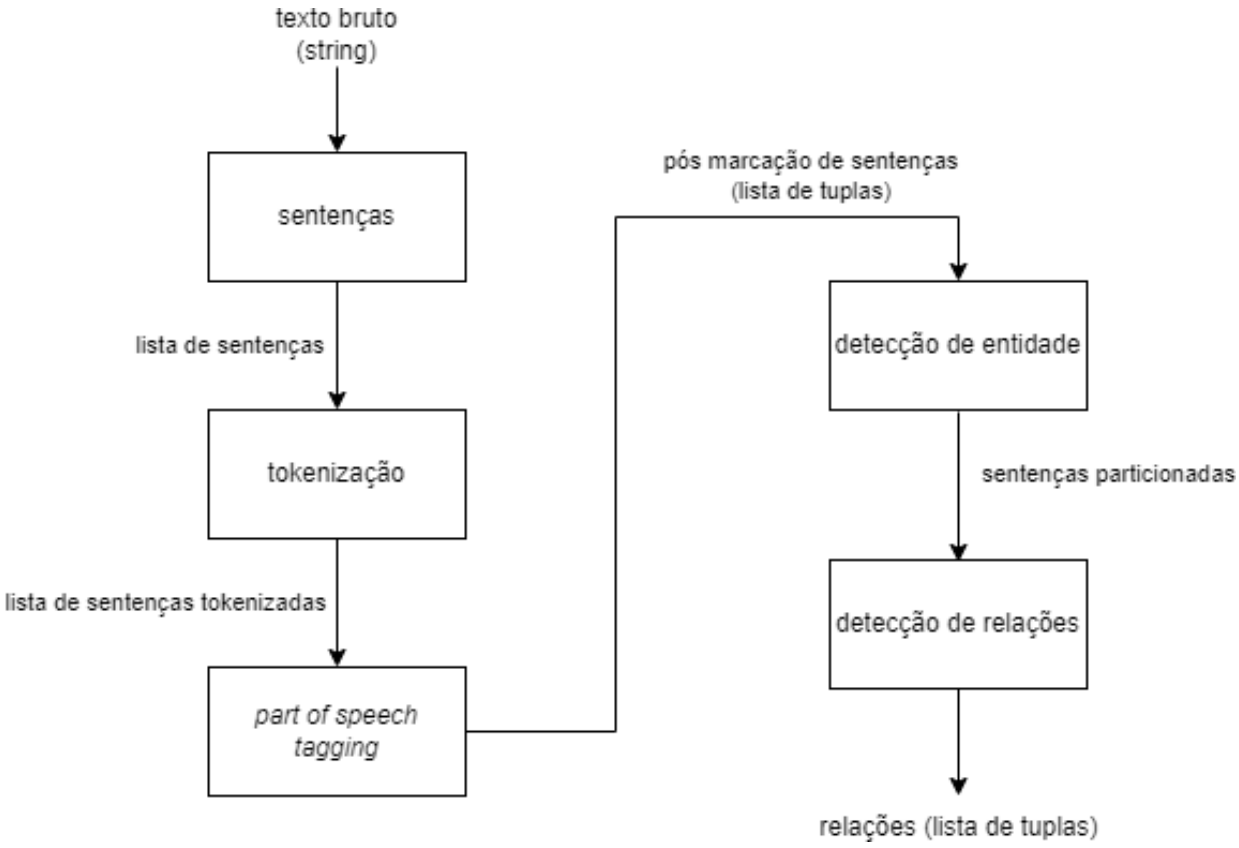
A ideia por trás do PLN, é dar aos computadores a capacidade de lidar com textos escritos por pessoas, isso inclui analisar o seu contexto, considerar diferenças de linguagem, retirar informações, entender os sentidos das frases e até compor ou recuperar textos em resposta (como é o caso a ser estudado no presente trabalho). De acordo com Neto, Tonin e Prietch (2010), o PLN permite que os seres humanos se comuniquem com os computadores da forma mais "natural"possível, utilizando a linguagem com a qual mais estão acostumados.

Conforme Goldberg (2017), são várias aplicações que podem ser desenvolvidas quando é possível transformar textos em dados estruturados, algumas delas são: classificação textual, linguística aplicada, assistentes pessoais, análise de mídias, robótica e IA genérica.

A aplicação de métodos de PLN, permite que o computador possa processar dados textu- ais com algoritmos específicos. Assim, é possível treinar modelos de aprendizado a partir desses dados. Esses modelos de aprendizado funcionam entendendo a língua humana e capacitando o computador a responder buscas, criar respostas por meio de textos que façam sentido com as interações de usuário, além de outras abordagens que podem ser feitas com esses modelos inte- ligentes.

Ao trabalhar com PLN há algumas etapas básicas, uma prática comum é dividir os pro- blemas em tarefas menores, conforme representadas na Figura 1: segmentação de sentenças, to- kenização, *Part-Of-Speech (POS)*, *tagging*, *chunking*, *parsing*, entre outras (ELHADAD, 2010). Rosa (2011) afirma que “As aplicações da IA vão desde jogos até prova de teoremas.

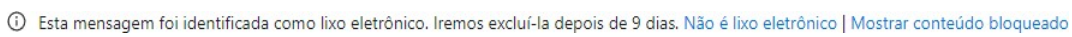
Figura 1 – Etapas básicas de PLN



Fonte: Adaptado de Rodríguez e Bezerra (2020).

Muitas das tarefas de que a IA trata podem ser divididas em tarefas cotidianas, tarefas formais e tarefas especialistas", entre outras aplicações, como o *chatbot*, que, de acordo com AbuShawar e Atwell (2015) é um agente de um software de conversação, que interage com usuários, usando a linguagem natural. O uso de PLN também é encontrado em filtros de e-mail, onde o sistema identifica se os e-mails pertencem a uma das categorias programadas e de acordo com seu conteúdo, reduz o número de e-mails na caixa de entrada principal, fazendo com que destaque apenas e-mails relevantes para o usuário (Figura 2). Outros exemplos presentes são a tradução de idiomas, sugestão de correção de textos, resultados de pesquisa (presente na Figura 3), onde apresenta uma busca semelhante já feita anteriormente ou de outros usuários.

Figura 2 – E-mail e identificação de SPAM

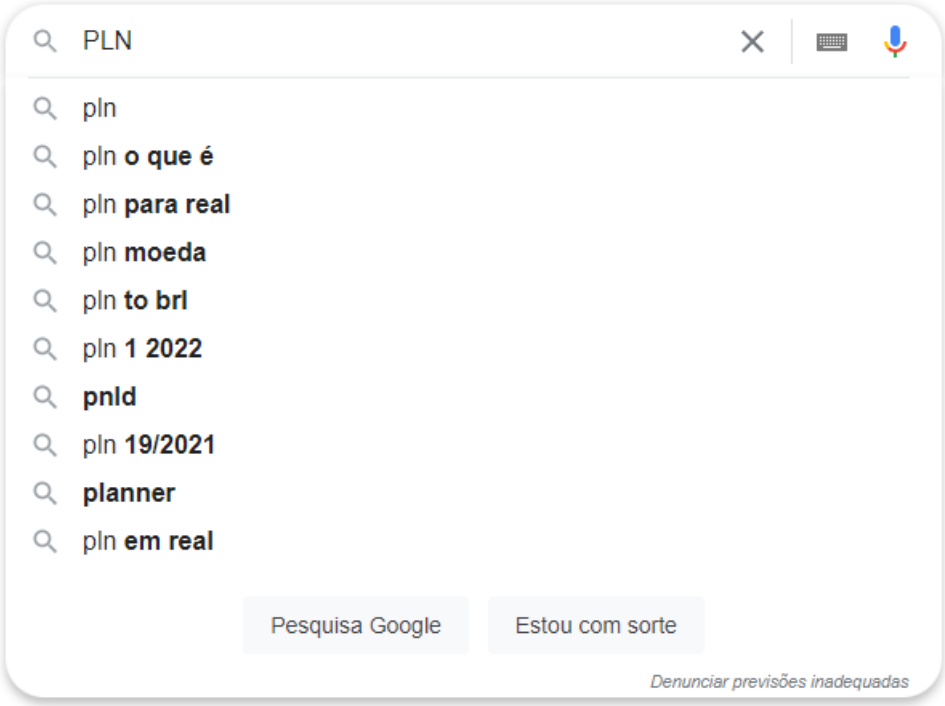


Fonte: Autoria própria (2022).

2.3 Pré-processamento

O objetivo do pré-processamento é de transformar os dados textuais, facilitando a aplicação de técnicas de mineração de textos, o objetivo dos métodos de refinamento de dados tex-

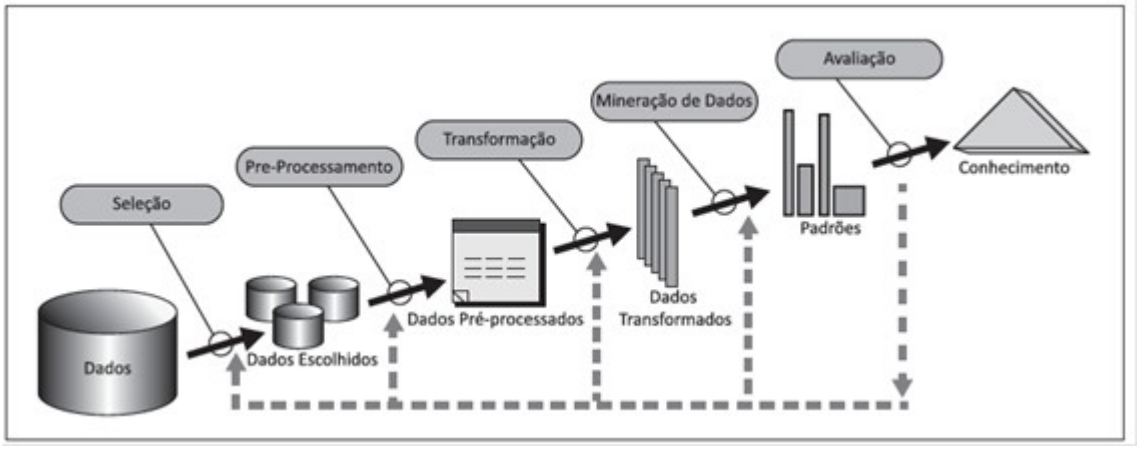
Figura 3 – Resultado de pesquisas com PLN



Fonte: Autoria própria (2022).

tuais é validar e aperfeiçoar o conhecimento adquirido (LIMA, 2017). O ciclo é representado na Figura 4.

Figura 4 – Pré-processamento



Fonte: LIMA (2017) e Han, Kamber e Pei (2011).

O pré-processamento consiste em transformar um texto bruto em uma estrutura de representação para os algoritmos usarem, com o principal objetivo de aumentar a qualidade dos dados obtidos. Considerada a etapa mais onerosa do processo, por não contar com uma única técnica, mas sim com várias técnicas.

Para que os dados sejam processados é necessário que eles sejam tratados, e que seja feita a preparação do texto para possibilitar a extração adequada de suas características, o pré-

processamento consiste na preparação desses dados (SILVA, 2014). Esse procedimento é neces- sário, pois a linguagem natural é conhecida por ter um grau de complicação, dessa forma são extraídas dos textos apenas as características relevantes, que devem ser transformadas em uma representação que permita ao computador processá-las.

É considerável citar que ao utilizar o PLN, há algumas técnicas, como remoção de pala- vras indesejadas (*stopwords*), remoção de sufixos (reduzindo uma palavra ao seu formato raiz), lematização (substituindo uma palavra flexionada por sua forma base), agrupamento de palavras múltiplas, normalização de sinônimo.

- Tokenização: A tokenização transforma os textos em sequências de *token*, conhecida como segmentação de palavras, quebra a sequência de caracteres em um texto, localizando o limite de cada palavra, ou seja, os pontos onde uma palavra termina e outra começa. Tokenização é bem estabelecida e bem entendida para linguagens artificiais tais como linguagens de programação (SMART, 2016). Para fins de linguística computacio- nal, as palavras assim identificadas são frequentemente chamadas de *tokens*. Um *token* corresponde a uma palavra ou termo, ou seja, o termo mais geral usado para incluir pon- tuação e outros símbolos (STRAKA; HAJIC; STRAKOVÁ, 2016).

- Normalização: A normalização é utilizada para padronizar todos os dados de entrada, co- locando todo o texto em letras minúsculas, removendo *tokens* indesejados como espaços em branco, caracteres especiais (ALVARENGA, 2019).

Quadro 1 – Exemplo de normalização

Antes	**Após a normalização e tokenização**
Esta é uma normalização e tokenização por PA-LAVRAS	["esta", "é", "uma", "normalização", "e", "tokeni-zação", "por", "palavras"]

Fonte: Autoria própria (2022).

- Remoção de *stopwords*: A remoção de *stopwords* nada mais é do que a remoção de pa-lavras consideradas irrelevantes, que não contribuem para o significado geral da frase (HARDENIYA *et al.*, 2016). Uma das tarefas mais utilizadas no pré-processamento de textos, que consiste em remover palavras muito frequentes, tais como “,”, “a”, “de”, “o”, “da”, “que”, “e”, “do”, “para, “com”, entre outras, pois na maioria das vezes não são informações relevantes para o desenvolvimento do trabalho e, portanto, podem ser remo- vidas.

2.4 Representação de Dados

A representação de dados, no quesito de máquinas, são representados através de números (linguagem numérica). Nesse sentido, é preciso transformar informações da linguagem humana, para a linguagem que o computador consiga entender.

No sentido de algoritmos de aprendizado de máquina, podemos dizer que processam entradas e saídas de comprimento fixo e bem definidas, não é interessante trabalhar diretamente com textos em original, já que tem dados textuais não padronizados. Para isso, é necessário que o texto seja convertido em uma representação numérica e vetorial de forma que estes dados se tornem viáveis de ser utilizados em um processamento computacional.

Uma das formas mais tradicionais de representação de textos é o formato vetorial, tam- bém conhecido como *Bag-of-word* (BOW). Não fora utilizado no presente trabalho por ter certas limitações (se comparado com o *Word Embedding*), como: vetores (resultados) obtidos teriam o mesmo tamanho do vocabulário, se tornando um problema com vocabulário grandes, e é co- mum em trabalhos, lidarmos com vocabulários com muitas palavras. Outra limitação é que eles não conseguem captar significado entre palavras, isto é, extraem pouca informação semântica e sintática dos textos.

Uma forma de representação bem atual, é o *Word Embedding*, uma técnica muito popular para representar um vocabulário predefinido e de tamanho fixo de palavras em documentos. Esta técnica é capaz de capturar o contexto de uma palavra em um documento assim como sua semelhança semântica, sintática, sua relação com outras palavras (MIKOLOV *et al.*, 2013).

Estudos e aplicações com *Word Embedding* têm sido aplicadas com sucesso e inume- ros problemas envolvem dados textuais (WANG *et al.*, 2016). *Word Embedding* são vetores dimensionais que relacionam uma palavra a um determinado contexto. Estes vetores otimizam o processamento computacional, tornando ações complexas de palavras em operações com ma- trizes mais simples (LEVY; GOLDBERG, 2014). A ideia por trás dessa técnica é que palavras num mesmo contexto tendem a ter significados semelhantes, postando-se então próximas num gráfico vetorial.

Os modelos de *embeddings* podem ser classificados em contextualizados e não-contextualizados. Os não-contextualizados, como o Word2Vec, um dos algoritmos mais popula- res para gerar *word embedding*s a partir de um corpus de texto, é um algoritmo de aprendizado não supervisionado, que tenta aprender automaticamente a relação entre palavras, agrupando palavras que tenham significados semelhantes em *clusters* semelhantes (RASCHKA, 2015). Sendo representado através de vetor n-dimensional, tendo a ideia que a distância entre os ve- tores que representam as palavras *Word Embedding* seja menor se as palavras correspondentes forem semanticamente mais semelhantes. Já o classificado em contextualização onde produz diferentes representações vetoriais para a mesma palavra em um texto, o que varia de acordo com o seu contexto e, portanto, são capazes de capturar semânticas contextuais de palavras ambíguas (AKBIK; BERGMANN; VOLLGRAF, 2019).

Dentre os modelos de *embeddings* contextualizados, tem-se o algoritmo BERT. É o algoritmo de aprendizado profundo da Google para PLN, o qual ajuda computadores e máqui- nas a entender a linguagem como os humanos. É um modelo de representação de linguagem projetado para pré-treinar representações bidirecionais profundas de texto não rotulado, condi- cionando conjuntamente no contexto esquerdo e direito em todas as camadas (DEVLIN *et al.*,

2018).

Em resumo, o BERT pode ajudar a entender melhor o significado das palavras nas consultas no mecanismo de busca. Vale lembrar que o BERT é um modelo pré-treinado (DEVLIN *et al.*, 2018) e de código aberto (*open-source*), que é utilizado para produzir *embeddings* con- textuais que podem ser usados para tarefas de aprendizado (não-supervisionado) e foi lançado em 2018.

O funcionamento do BERT consiste basicamente em fornecer um modelo que, dada uma frase completa (nesse contexto chamada de sentença), irá gerar um *word embedding* para cada palavra dentro do contexto da frase.

O BERT é baseado na arquitetura do modelo *Transformer*. Em sua forma original, o *Transformer* inclui os dois mecanismos separados, um codificador e um decodificador. O uso do *Transformer* pode ser ilustrado com um exemplo de tradução do francês para o inglês, conforme demonstrado na Figura 5 (CHAVES, 2021).

A arquitetura do *Transformer* é um tipo de rede encoder-decode, inicialmente pensada para a tarefa de tradução (VASWANI *et al.*, 2017). O objetivo principal desse tipo de rede é permitir uma paralelização maior por depender inteiramente de mecanismos de atenção, em vez de utilizar modelos recorrentes que são intrinsecamente sequenciais (SOUZA; NOGUEIRA; LOTUFO, 2019).

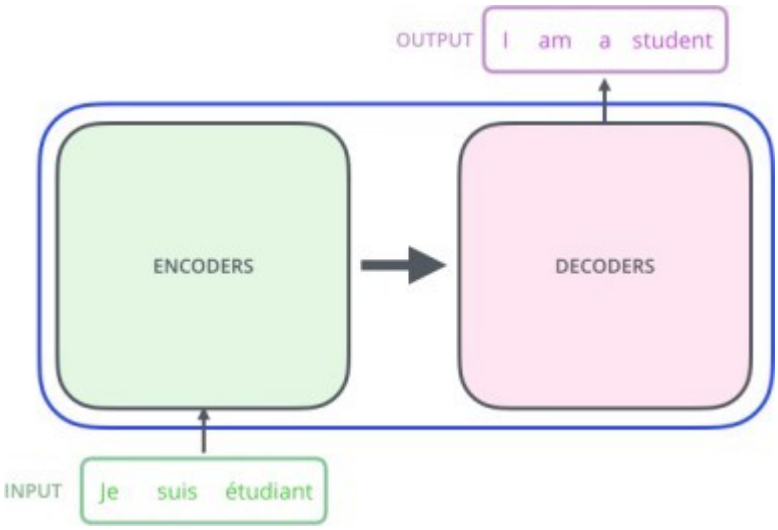
O modelo de algoritmo baseado em Mecanismo de atenção são vários, mas os destaque são o *Transformer* (VASWANI *et al.*, 2017) e o BERT (DEVLIN *et al.*, 2018) (sendo o BERT utilizado nesse trabalho), ambos são algoritmos complexos, cujas representações envol- vem muitos detalhes, mas baseado no mecanismo de atenção. Mecanismo de atenção permitem uma consideração maior ao contexto de representação textual (SERRAS, 2021), ele permite não só traduções mais precisas, mas melhores resultados para qualquer tarefa que necessite “tradu- zir” uma sequência em outra, seja por exemplo: transformando um texto em seu resumo ou uma imagem em uma legenda.

2.5 Aprendizado de Máquina

Para Lary *et al.* (2016) a Aprendizagem de Máquina vem provando ser útil em diversas aplicações relacionadas à geociência, mas também há diversos tipos de aplicações para aprendi- zado de máquina, como por exemplo: detecção de defeitos em software (ALEEM; CAPRETZ; AHMED, 2015), previsão de comportamentos anômalos a partir de uma base de dados (OMAR; NGADI; JEBUR, 2013), categorização de tweets em categorias de Segurança Pública (SAN- TOS, 2015), já na área da saúde, aplicação de conceitos e técnicas estão servindo como auxílio em tarefas relacionadas ao câncer, sendo um dos temas recorrente em estudos recentes, estima- se que mais de 500 publicações acadêmicas em periódicos médicos são geradas a cada ano (GANT; RODWAY; WYATT, 2001).

Aplicações de aprendizado de máquina são encontrados no dia a dia, como nos e-mails,

Figura 5 – Encoder-decoder aplicado a um problema de tradução. Os *embeddings* de interesse situam-se na interface entre o encoder e o decoder.



Fonte: Chaves (2021).

na detecção de Spam automaticamente, nas compras na internet, quando se pesquisa um produto e aparece vários, ou quando se está vendo um produto e surge outros semelhantes. Estas são alguns dos casos que estão presentes, mas não é percebido como acontece.

A seguinte seção apresenta revisões fundamentais sobre aprendizado de máquina, dando ênfase em conceitos, categorização, algoritmos e aplicações existentes.

2.5.1 Conceitos sobre Aprendizado de Máquina

O aprendizado de máquina, do inglês, *machine learning* é uma subárea da IA e sub-campo da ciência da computação, que tem como objetivo criar técnicas computacionais e siste- mas que, automaticamente, adquirem conhecimento. É uma evolução do estudo de reconheci- mento de padrões e da teoria do aprendizado computacional em IA.

Ao questionar se a máquina seria tão boa quanto um humano, é notável ressaltar que não é possível fazer algoritmos/computador/máquina aprenderem tão bem quanto as pessoas, mas diversos algoritmos criados são eficientes para várias tarefas de aprendizado, e os estudos teóricos e pesquisas sobre aprendizado estão permitindo que novas técnicas sejam desenvolvidas (MITCHELL, 2009).

É uma ferramenta de aquisição automática de conhecimento, a técnica de aprendizado de máquina tem diversos algoritmos com diferentes desempenhos, de acordo com a necessidade cada problema, sendo importante compreender suas limitações de desempenho. No contexto em que se insere, o aprendizado se refere à inferência indutiva (diz respeito partir de um caso espe- cífico para deduzir um caso geral) (RÄTSCH, 2004), abordado na categorização dos mesmos, conforme apresentado na subseção 2.5.2.

2.5.2 Algoritmos de aprendizado

Os algoritmos de aprendizado podem ser divididos em diferentes categorias, sendo:

- **Aprendizado supervisionado**: O aprendizado supervisionado consiste no conjunto de técnicas em que existe um "supervisor". Assim, quando um conjunto de exemplos rotulados é fornecido, e baseado neste conjunto, o algoritmo de aprendizado escolhido generaliza para responder corretamente a todas possíveis entradas futuras. Para efetuar o aprendizado, o algoritmo com sua saída com a deste supervisor, para que assim, se ajuste em busca de minimizar o erro de sua resposta (BISHOP, 2006). Para esse conjunto de exemplos rotulados dá-se o nome de conjunto de dados de treinamento, pois é a partir desses exemplos que o algoritmo será treinado para execução de futuras generaliza- ções. Alguns algoritmos de aprendizado que podem ser citados são: Análise de Regressão Linear, Análise de Regressão Logística, *NAIVE BAYES, SUPORT VECTOR MACHINE (SVM), K-NEAREST NEIGHBORS (KNN)*, dentre outros. Tarefas de classificação, esti- mação e predição estão associadas ao método de aprendizagem supervisionado conforme (CAMILO; SILVA, 2009).

- **Aprendizado não-supervisionado**: Nesta categoria, não há um resultado alvo. Os algoritmos agruparão um conjunto de dados em diferentes grupos. Os algoritmos não supervisionados, não precisam ser treinados com dados rotulados (SCHMITT, 2013). As respostas corretas para cada um dos objetos de entrada não são fornecidas, por isso os al- goritmos procuram identificar similaridades entre os objetos de entrada, uma ferramenta utilizada nesse aprendizado é a análise de *cluster* e para cada grupo, ou *cluster*, um ró- tulo é utilizado, permitindo indicar a qual grupo cada registro pertence (DUDA; HART; STORK, 2001). são exemplos de algoritmos dessa classe: K-MEANS, C-MEANS, PCA, entre outros.

- **Aprendizado por reforço**: Segundo Marsland (2011), pode haver mais uma classificação, a do aprendizado por reforço. Esses algoritmos são treinados para tomar decisões. Portanto, com base nessas decisões, o algoritmo treinará com base no sucesso/erro de saída, ou seja, Marsland ainda complementa, dizendo que a máquina aprende por meio da interação com o ambiente. Eventualmente, o algoritmo de experiência poderá dar boas previsões, é trabalhado com a ideia de condicionamento (saber se uma ação é correta ou não) a partir de recompensas ou punições. Segundo a SAS (2021), a aprendizagem por reforço é composta por três componentes principais: o agente (Tomador de decisões); o ambiente (Tudo com o que o agente interage); as ações (O que o agente vai fazer).

Algoritmos de *machine learning* são uma ferramenta para aquisição de conhecimento, a partir da experiência, a utilização dos mesmos permite avanços e descobertas que garantem uma vantagem competitiva para as empresas que utilizam.

Dentre os algoritmos supervisionados mais populares, se encontram: regressão linear e suas técnicas (ex. *Multiple Linear Regression, Linear Regression, Elastic Net Regression, Ada Boosting Regression, Gradient Boosting Regression* (MADHURI; ANURADHA; PUJITHA, 2019)); árvores de decisão (NORVIG; RUSSELL, 2013); e Naive Bayes (ZEVAREX; SANTOS, 2020).

Existem diversos algoritmos de *clustering*, que usam outros esquemas de agrupamento e se diferem em desempenho, medidas de similaridade ou dissimilaridade, seleção ou extração de recursos, validação do *cluster* e complexidade (BINDRA; MISHRA, 2017).

Clustering (do Português Agrupamento), é uma técnica de aprendizagem de máquina não-supervisionada (BELTRAN, 2019), em que se busca agrupar os pontos de dados com base em características específicas. Há vários algoritmos de agrupamento que usam essa técnica, como o K-Means e o próprio DBSCAN.

A seguir são apresentados alguns dos algoritmos de *clustering* mais populares:

- *K-means*: um método de aprendizagem de máquina não-supervisionado, onde sua função é realizar a aproximação de pontos de dados similares ou mais próximos, gerando *clus- ters*, fazendo com que *clusters* vizinhos não apresentem similaridade entre si (MARS- LAND, 2011).

- *C-means*: um método de agrupamento difuso que permite que um objeto pertença a dois ou mais *clusters* com grau de associação entre um e zero (PACHECO, 2018). E da mesma forma que o K-means, este algoritmo utiliza a distância Euclidiana para encontrar *clusters* de forma circular. No entanto, diferente do DBSCAN, o algoritmo possui dificuldades em lidar com dados ruidosos, e algumas limitações na identificação de *clusters* de diferentes formas (PACHECO, 2018).

- DBSCAN: proposto por (ESTER *et al.*, 1996), é um método de agrupamento baseado na densidade de dados, seu objetivo é encontrar concentrações de elementos que estão espa- cialmente próximos. São buscados pontos que possuam um número mínimo de vizinhos (*min_points*) dentro de um certo raio (*eps*). Quando os pontos de um *cluster* alcançam pontos de outro *cluster* (pontos vizinhos). Isso acontece repetidamente enquanto houver pontos alcançáveis, dessa forma são criadas regiões (BELTRAN, 2019) densas, caracte- rizando os *clusters*.

2.6 DBSCAN

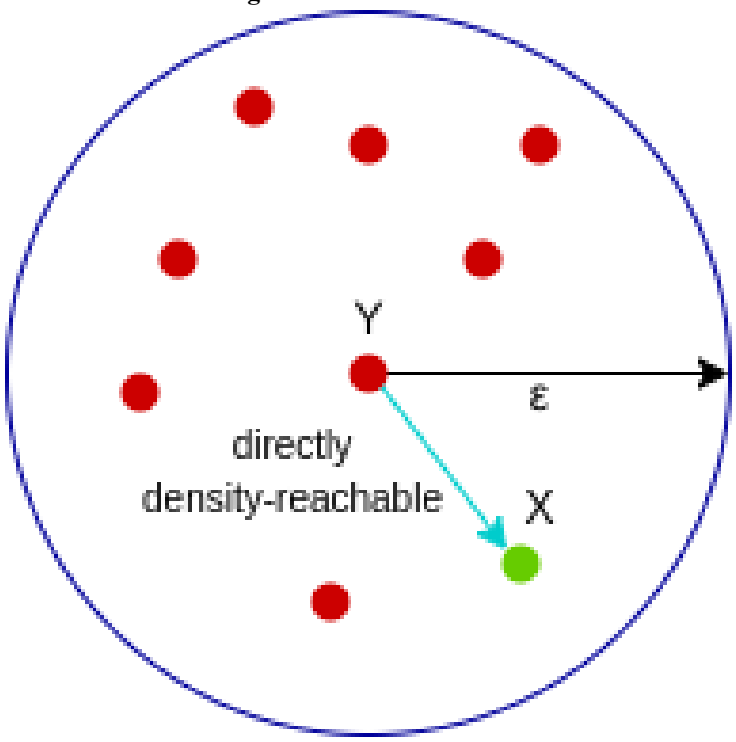
O DBSCAN é um algoritmo de *clustering* baseado em densidade. A principal característica desse algoritmo é que um *cluster* é definido por um raio (*eps*) e a vizinhança de cada ponto em um *cluster* deve ter um número mínimo de pontos (*min_Pts*) (PUPPO, 2021). Este

algoritmo provou ser extremamente eficiente na detecção de *outliers* e no tratamento de ruídos. Além disso, se comporta bem para uma base de dados bastante volumosa.

A escolha desse algoritmo para este estudo ocorreu, pois se destaca como um dos mais utilizados, além de que este algoritmo apresenta importantes vantagens em relação aos demais algoritmos de agrupamento, tais como: se destaca por ser capaz de encontrar *clusters* com formatos arbitrários, além de ser capaz de lidar com ruídos nos dados, característica que não está presente na maioria dos algoritmos de agrupamento (NETO, 2016).

A Figura 6 representa um resultado gráfico do DBSCAN através d e um conjunto de dados com *clusters* baseados em densidade. Desta forma, pode-se observar que o algoritmo retorna os seguintes pontos:

Figura 6 – DBSCAN *clusters*.



Fonte: Yang *et al.* (2020).

Um ponto X é diretamente alcançável por densidade a partir do ponto Y (conforme o minPoints parametrizado) se:

- X pertence à vizinhança de Y , ou seja, dist(X, Y) <= epsilon *(eps)*; E Y é um ponto central.

- Ponto central: é um ponto central, se houver pelo menos *min_Points* em sua vizinhança, ou seja, dentro do raio (*eps*) especificado (GAVA *et al.*, 2013).
- Ponto de fronteira: um ponto de dados é classificado como ponto de fronteira se: sua vizinhança contém menos *min_Points* pontos de dados, ou é alcançável a partir de algum ponto central, ou seja, está dentro da distância *EPS* de um ponto central (CASSIANO; PESSANHA, 2014).

- Ponto *outlier*: Um *outlier* é um ponto que não é um ponto central e, também, não está próximo o suficiente para ser alcançado a partir de um ponto central (VIBRANS *et al.*, 2012), é um ponto fora da curva, pode-se considerar um ruído.

Os outros pontos discrepantes são eliminados. Os pontos principais e que são vizinhos são conectados e colocados no mesmo *cluster*. Os pontos de fronteira são atribuídos a cada *cluster* com base na sua distância do ponto central (TAVARES, 2009).

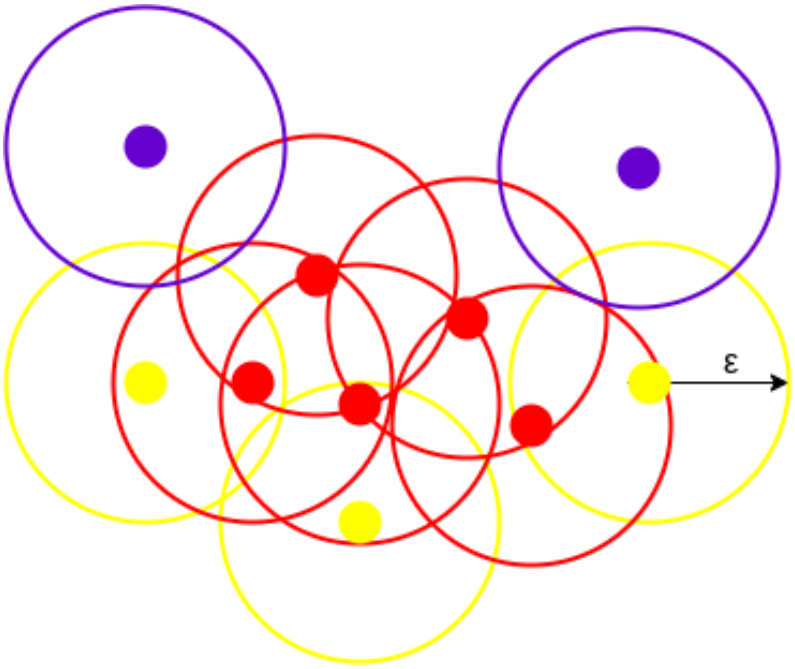
O DBSCAN requer dois parâmetros: raio (*eps*) e o número mínimo de pontos necessários para formar uma região densa (*min_Pts*). Tendo um ponto de partida para ínicio, a vizinhança deste ponto é selecionada e, se contiver pontos suficientes, um *cluster* é iniciado. Caso contrário, o ponto é rotulado como ruído (SCHUBERT; HESS; MORIK, 2018).

Se um ponto é uma parte densa de um *cluster*, sua vizinhança também faz parte desse *cluster*. Assim, todos os pontos encontrados dentro da vizinhança são somados, assim como sua própria vizinhança quando também são densas. Este processo continua até que o *cluster* conectado por densidade seja completamente encontrado. Então, um novo ponto não visitado é recuperado e processado, levando à descoberta de mais um *cluster* ou ruído (SCHUBERT; HESS; MORIK, 2018).

DBSCAN pode ser usado com qualquer função de distância (assim como funções de similaridades) (ESTER, 1996) e (SCHUBERT *et al.*, 2017). O algoritmo pode ser expresso em pseudocódigo da seguinte forma:

Um exemplo é apresentado na Figura 7, nos mostra um *cluster* criado pelo DBCAN com *min_Pts* = 3. Aqui, desenhamos um círculo de raio igual épsilon ao redor de cada ponto de dados. Esses dois parâmetros ajudam na criação de *clusters* espaciais. Todos os pontos de dados com pelo menos 3 pontos no círculo, incluindo ele mesmo, são considerados pontos. Todos os pontos de dados com menos de 3 mas maiores que 1 ponto no círculo, incluindo ele mesmo, são considerados como pontos de fronteira. Eles são representados pela cor amarela. Por fim, os pontos de dados sem nenhum ponto além de si mesmo presentes dentro do círculo são considerados como Ruído representado pela cor roxa.

Figura 7 – Exemplo de *clustering* com DBSCAN.



Fonte: Yang *et al.* (2020).

Figura 8 – Pseudo-código - DBSCAN.

```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N := RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) := Noise                               /* Label as Noise */
            continue
        }
        C := C + 1                                          /* next cluster label */
        label(P) := C                                       /* Label initial point */
        SeedSet S := N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point Q */
            if label(Q) = Noise then label(Q) := C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed (e.g., border point) */
            label(Q) := C                                   /* Label neighbor */
            Neighbors N := RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check (if Q is a core point) */
                S := S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

Fonte: Schubert *et al.* (2017).

2.7 Métodos de Redução de Dimensionalidade

A redução da dimensionalidade também é benéfica uma vez que tende a reduzir o supe- rajustamento (*overfitting*), muitos agrupadores apresentam baixo desempenho quando manipu- lam uma grande quantidade de atributos (SCHNEIDER, 2004). Dessa forma, é recomendável um procedimento para reduzir o número de termos utilizados para representar as mensagens (FORMAN *et al.*, 2003) e (FORMAN; KIRSHENBAUM, 2008).

PCA: permite reduzir a complexidade de um conjunto de dados e determinar informações sobre o relacionamento interno dos dados do conjunto. O uso de um algoritmo de PCA Robusto permite que se elimine a influência de dados dispersos na classificação (RAZERA; MACIEL, 2010). O objetivo principal do PCA é a redução de dimensionalidade de um conjunto de dados, cujas variáveis estão inter-relacionadas, e manter o máximo de variância presente no conjunto de dados (CADIMA; JOLLIFFE, 2009). O exemplo presente na Figura 9 do Vander- Plas (2016), onde os pontos claros são os dados originais, enquanto os pontos escuros são a versão projetada. Isso deixa claro o que significa uma redução de dimensionalidade do PCA: as informações ao longo do eixo ou eixos principais menos importantes são removidas, deixando apenas os componentes dos dados com a maior variância. A fração de variância que é cortada (proporcional à dispersão dos pontos sobre a linha formada nesta figura) é aproximadamente uma medida de quanta "informação"é descartada nessa redução de dimensionalidade.

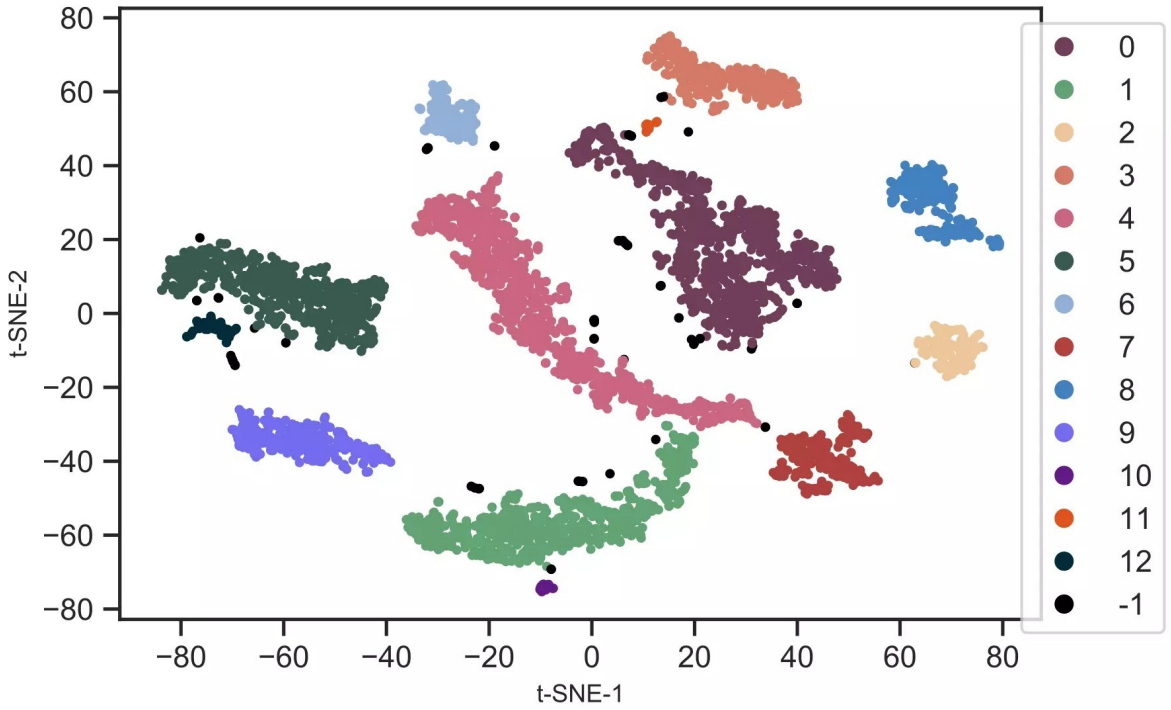
Figura 9 – Exemplo de PCA.

Fonte: VanderPlas (2016).

TSN-e: O algoritmo TSN-e fornece um método eficaz para visualizar um conjunto com- plexo de dados. Ele descobre com sucesso estruturas ocultas nos dados, expondo *clusters* na-

turais e suavizando variações não-lineares ao longo das dimensões (FILHO, 2019). O exemplo presente na Figura 10, onde os pontos dentro dos *clusters* individuais são altamente semelhantes entre si e distantes de pontos em outros *clusters*. O mesmo padrão provavelmente se aplica a um conjunto de dados original de alta dimensão. No conjunto de dados de dígitos, o TSN-e separou os *clusters* de cada classe de dígitos Xiang *et al.* (2021).

Figura 10 – Exemplo de TSN-e.



Fonte: Xiang *et al.* (2021).

2.8 Sistemas Existentes para agrupamento de *Bugs* de Software

A partir de estudos sobre Aprendizado de Máquina e PLN, pode-se perceber que com o agrupamento de *bugs* de software desenvolvido a partir desses métodos (e Aprendizagem de Máquina supervisionada e PLN), é possível reduzir a dependência humana no processo de agru- pamento dos chamados e atividades de priorização de atendimento, seleção de técnico adequado para o atendimento, e listagem de possíveis soluções, com dados localizados por meio dos do serviços de TI (*help-desk*).

Já existem alguns experimentos realizados com técnicos e o *help-desk*, como é o caso do trabalho de Ellen e Moura (2016), onde fora executado o agrupamento semiautomático de servi- ços de TI registrados em linguagem natural, com a realização de um experimento com mais de
2.000 chamados de *help-desk*, distribuídos em classes, previamente anotadas por especialistas em *service desk*. Na classificação eles utilizaram os algoritmos Naive Bayes e J48, o qual con- forme Patil (2013) diz, é baseado em árvore de decisão, e permite derivar regras de produção,

de decisão ou de classificação com a possibilidade de visualização em árvore dos resultados. Além desses desses algoritmos foi aplicado o SMO (PLATT *et al.*, 1999), explica que o SMO lida particularmente bem com conjunto de dados esparsos, característica detectada na matriz de *features* gerada pelo modelo proposto e utilizada como entrada dos algoritmos de Aprendiza- gem de Máquina), onde obtiveram a porcentagem de exatidão nos resultados de 72.9%, 70.7% e 75.8%, respectivamente.

Outro caso de aplicação de PLN em sistemas de *help-desk* foi realizado pelo Manhães (2014), em que a partir de uma classificação baseada no Classificação Ortogonal de Defeitos (ODC) e em *Information Technology Service Library* (ITIL), puderam ser realizados vários trabalhos objetivando melhorias na extração dos dados antes de classificá-los, com a finalidade de classificar a resolução de defeitos em manutenção de software. Sendo assim, propuseram a separação de forma clara nos incidentes ao realizar a abertura e o fechamento dos chamados de usuários, para uma montagem do *data warehouse* de suporte de incidentes com as soluções associadas à classes de problemas. Para isso foram utilizados métodos de mineração de texto e comparações entre diversos algoritmos. Contudo, a base de conhecimento utilizada nas soluções associadas a cada classe de incidentes, em muitos casos, não foram suficientes para diminuir o tempo de atendimento a problemas mais complexos, pois detalhes da resolução não foram bem especificados. Ainda segundo Manhães (2014), houve uma dificuldade em classificar os defeitos em relação ao número e tipo de defeitos. O que implica que a maioria das definições usadas faz com que não haja garantia de que dois diferentes indivíduos, olhando para o mesmo conjunto de falhas e a para as mesmas definições de falhas, farão uma classificação similar.

O que se afirma é a dificuldade presente nas classificações, que já havia sido citado por Siekmann *et al.* (1999), abordagens puramente linguísticas também encontram dificuldades pois o conhecimento linguístico é muitas vezes aplicável em domínios muito específicos, não sendo portável à outros domínios. Além disso, as técnicas linguísticas (como etiquetagem de catego- rias gramaticais, resolução de ambiguidade, análise sintática, outras) são bastante complexas e necessitam ter alto grau de precisão para trazer benefícios. É valido citar que há uma escassez de bases de dados textuais voltadas para o domínio da engenharia de software, especialmente na língua portuguesa.

3 MATERIAIS E MÉTODO

A seguir estão o método e os materiais utilizados para a realização deste trabalho.

3.1 Materiais

Os materiais, incluindo ferramentas e tecnologias aplicadas ao desenvolvimento deste trabalho estão listadas na Quadro 2.

Quadro 2 – Tecnologias utilizadas para o trabalho

Ferramentas/Tecnologias	**Versão**	**Finalidade**
Python	3.9.5	Linguagem de programação
Google Colab	2021	Servidores da Google (máquina virtual), que permite escrever e executar código em Python no na- vegador de forma rápida e sim- ples
Visual Paradigm	16.3	Ferramenta para modelagem do sistema
NLTK (Natural language Tool-kit	3.6.2	Conjunto de bibliotecas e progra- mas para processamento simbó- lico e estatístico da linguagem na- tural
Pandas	1.2.5	Biblioteca de software criada para a linguagem Python com a finalidade de fazer manipulação e análise de dados
Scikit-learn	0.24	Biblioteca de aprendizado de má- quina para a linguagem de pro- gramação Python
Spacy	2.2.0	Biblioteca de software para pro-cessamento avançado de linguagem natural
BERT	2022	Modelo de linguagem pré-treinado de aprendizado profundo (*Deep Learning*) do Google para PLN.
Framework Sentence_Transfor-mer	2.2.2	*Framework* em Python para incorporação de frases, textos e imagens. Calcula *embeddings* de frases/textos para mais de 100 idiomas Reimers e Gurevych (2019). Desenvolvido por Devlin *et al.* (2018).
DBSCAN	2022	Algoritmo de agrupamento não paramétrico baseado em densidade.

Fonte: Autoria própria (2022).

A linguagem de programação escolhida foi o Python, pois em se tratando de PLN há inúmeras bibliotecas e estatísticas disponíveis nessa linguagem e que são muito acessíveis e eficientes. Algumas bibliotecas são únicas, como o Pandas que traz os poderosos "quadros de dados"para o Python. Para PLN, o Python ainda dispõe do *Natural Language Toolkit* (NLTK), uma biblioteca diferenciada para esse tipo de aplicação. As principais bibliotecas utilizadas na implementação são descritas abaixo:

- NLTK: Com PLN, há uma ferramenta com diversos recursos voltados especificamente para isso, que é a biblioteca NLTK. Dentre os diversos recursos oferecidos, vale citar inicialmente as funções de tokenização e radicalização. Ambas as técnicas são muito importantes para o processamento de textos.
- Pandas: Uma forma de manipular dados de maneira mais visual e simples, com mais recursos disponíveis para essa manipulação, é utilizando a poderosa biblioteca Pandas. Com este recurso em Python, facilita-se o acesso a várias formas de análise e estruturação de dados.
- *Scikit-Learn*: uma biblioteca voltada para o aprendizado de máquina, com diversos méto- dos de classificação/agrupamento disponíveis, além de métricas de avaliação em IA.

O Google Colab é uma espécie de máquina virtual, que não requer configuração ou ins- talação de bibliotecas e é executado na nuvem. Essa plataforma permite escrever e executar códigos em Python, assim como acessar poderosos recursos de computação, e tudo gratuita- mente. Sendo assim, com o Google Colab é possível implementar e executar todos os testes propostos, fazendo uso da linguagem de programação Python que contém bibliotecas essenci- ais para o PLN. Além disso foram utilizados alguns diagramas da *Unified Modeling Language* (UML) para modelar os processos envolvidos nesse estudo.

O modelo pré-treinado utilizado foi o BERT. A escolha desse modelo ocorreu, por ser um modelo de aprendizado profundo que utiliza o mecanismo de *selfattention* (VASWANI *et al.*, 2017), o que o torna um modelo de aprendizado contextualizado. BERT utiliza vetores de sub-palavras e o mecanismo de *Transformers*. Esse mecanismo permite analisar o contexto de cada palavra em um texto de forma individual, o que permite verificar se cada palavra já foi utilizada anteriormente em um mesmo contexto. Com isso, o método permite aprender relações contextuais entre palavras (ou subpalavras) em um texto, ou seja, uma forma mais completa se comparado ao nao contextualizado. Os *Transformers* não utilizam direcionalidade, ao invés disso eles utilizam o contexto inteiro da sentença de uma só vez (ANCHIÊTA *et al.*, 2021). Para mais detalhes sobre o BERT, ver seção 2.4. O modelo pré-treinado utilizado nos experimentos desse trabalho foi o *BERT_base*, pois a chamada do método do BERT está *base-uncased* isso faz ele ser *bert_base*, se fosse outro método (*large*) seria *large-uncased*.

O método escolhido foi o DBSCAN, pois ele permite agrupar dados, fazendo com que o uso do BERT, junto ao DBSCAN nos retorne a lista de *bugs* que já fora solucionado. Ainda

segundo Daniel (2016), o DBSCAN tem um diferencial, que não há necessidade de se ter um conhecimento preliminar dos dados analisados.

3.2 Método

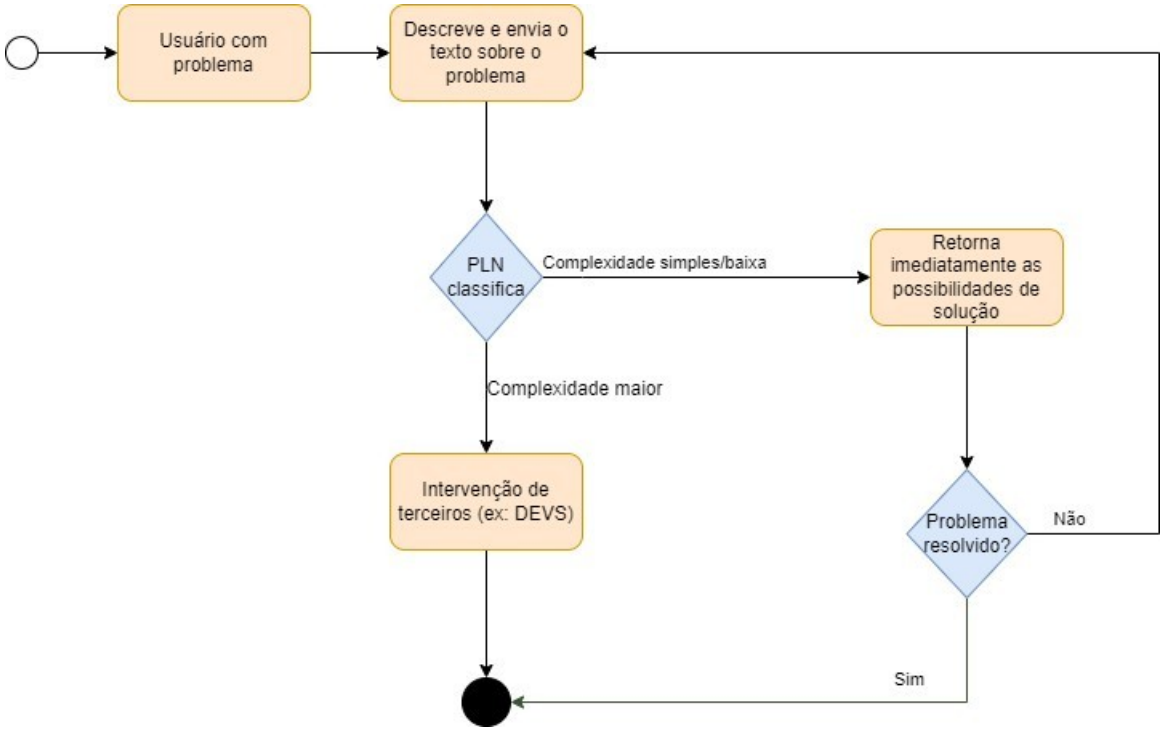
Normalmente, ao se deparar com um *bug* (ex. dificuldades de interação, erros inespera- dos, desconhecimento de uma funcionalidade, entre outros), os usuários entram em contato com o suporte, seja por telefone, e-mail ou mesmo um sistema de *help-desk*. Pode-se observar ainda, que uma das principais formas de interação do usuário com o suporte é por meio de textos que descrevem o seu problema. Sendo assim, esse trabalho objetiva apresentar uma metodologia para o problema do agrupamento de *bugs* de software, com base na exploração de textos de *bugs*, os quais já foram resolvidos no passado. Além disso, esse trabalho pretende criar uma base de dados de *bugs* de software na língua portuguesa.

Dentro desse contexto, esse trabalho apresenta a aplicação de métodos de PLN, aliados a métodos de aprendizado de máquina, a fim de gerar o agrupamento dos *bugs* apresentados. O objetivo do agrupamento é retornar uma listagem dos *bugs* mais similares ocorridos em situações anteriores e suas respectivas soluções possíveis. Desta forma, a metologia proposta poderá ser aplicada futuramente na implementação de um sistema de *help-desk* completo, no qual o usuário poderá receber um retorno imediato de possibilidades de solução para seu problema. Um exemplo de caso de utilização de um sistema como esse é apresentado no fluxo da Figura 11, em que um problema pode ser considerado de baixa complexidade ou simples (retornando uma lista de possíveis soluções, sendo o encaminhamento dado pela pessoa do suporte) ou de complexidade maior (fazendo uso da lista de possíveis soluções, mas requerendo intervenção de terceiros, por exemplo, da equipe de desenvolvimento).

Para a representação textual dos textos de *bugs*, será aplicado o modelo de linguagem contextutalizado BERT proposto por (DEVLIN *et al.*, 2018). Importante destacar que não há nenhum trabalho relacionado que tenha utilizado o modelo pré-treinado BERT como método de representação textual.

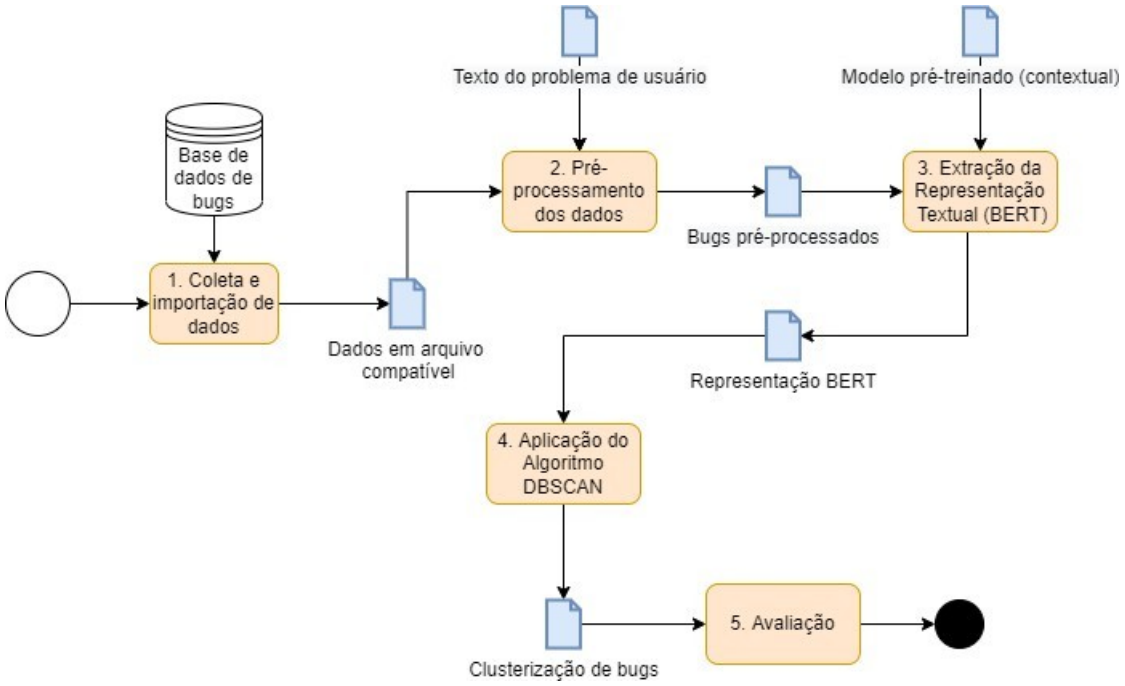
A seguir são apresentadas as etapas da metodologia adotada para o desenvolvimento deste trabalho (Figura 12), tais como: coleta de dados e importação dos dados, pré-processamento, extração da representação textual (BERT), método de *clustering* (DBSCAN) e avaliação.

Figura 11 – Fluxo de agrupamento de *bugs*



Fonte: Autoria própria (2022).

Figura 12 – Metodologia proposta

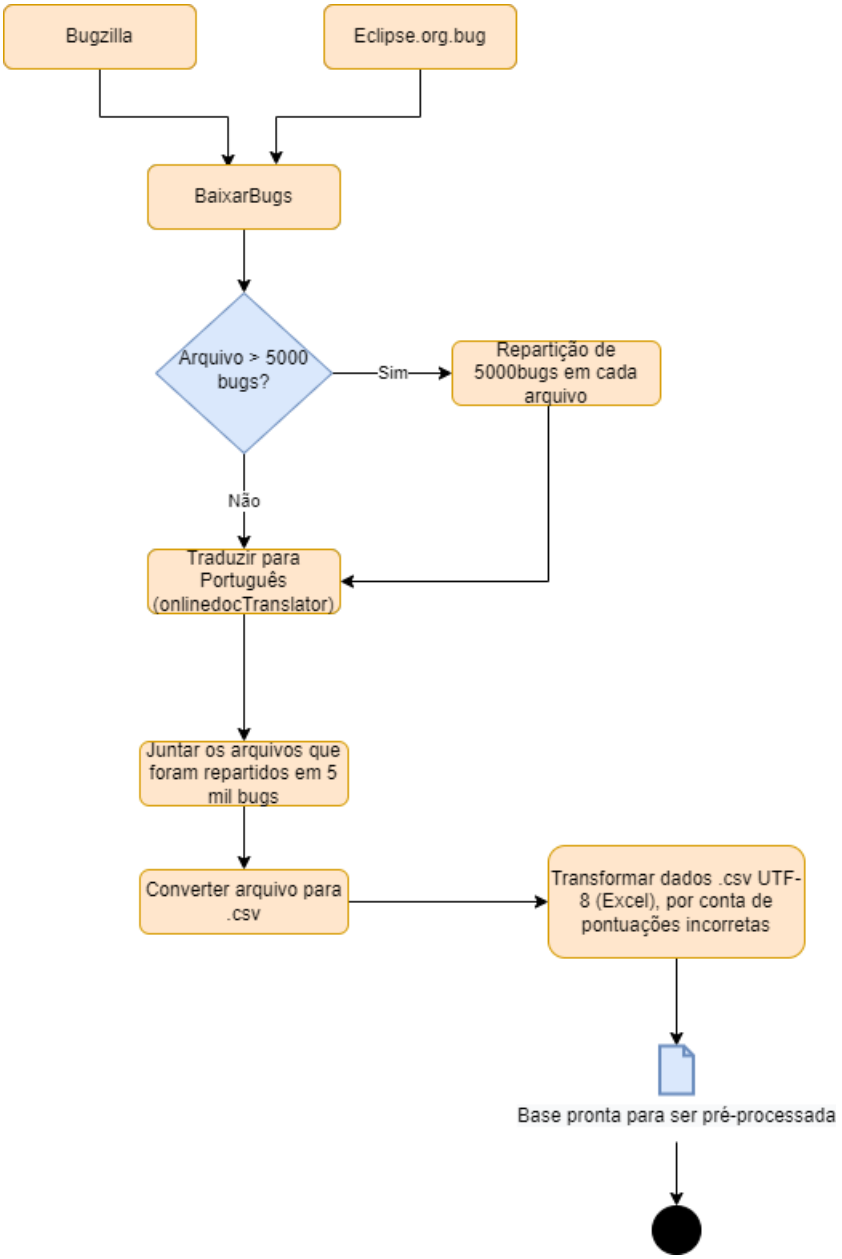


Fonte: Autoria própria (2022).

1. Coleta de dados: A primeira etapa a ser realizada é a coleta de dados, que se trata de um processo que pode ser custoso e burocrático, podendo ser realizada de diferentes formas, dependendo da natureza dos dados (MELO, 2010). Para isso, depois de muitas buscas na Internet, foi selecionada uma base de dados de *bugs* obtido nas ferramentas online para o gerenciamento de *bugs* dos projetos BugZilla (Figura 14) e Eclipse. Foram ne- cessários procedimentos para importar esses dados, conforme apresentados na Figura 13. Importante ressaltar que não há nenhuma base de dados textuais na área de Engenharia de Software, muito menos para *bugs* de software na língua portuguesa.

   Portanto, trata-se de uma etapa crucial para o resultado da aplicação proposta, pois a quantidade e a qualidade dos dados influenciam fortemente a qualidade dos resultados.

Figura 13 – Obtenção da base de dados



Fonte: Autoria própria (2022).

2. Pré-processamento dos dados: Após realizar a coleta dos dados (representado na Figura 13), eles devem passar por algumas etapas de pré-processamento. Para isso, foram aplicados alguns métodos, tais como: tokenização, remoção de caracteres especiais e remoção de *stopwords* (ex. *tags* HTML/XML), se esse for o caso.

3. Extração de características/representação textual: Essa etapa consiste na extração de características dos textos de *bugs*, que serão usadas como entrada no método de agrupamento a ser aplicado. A extração de características nesse trabalho ocorreu por meio da aplicação do modelo pré-treinado BERT, para a gerar a representação textual dos *bugs* de software no formato de *embeddings*.

4. Método de *clustering*: Após coletar e pré-processar os dados, ocorre a aplicação do método de aprendizado a ser utilizado, de acordo com as características dos dados disponíveis. Nesse caso, foi aplicado um método de *clustering*, o que se justifica, pois trata-se de um algoritmo de aprendizado não-supervisionado, em que os dados textuais utilizados não são rotulados. Desta forma, foi utilizado o DBSCAN pois é um método específico para agrupamento de dados e representa algumas vantagens aos outros algoritmos como o K-means, como identificação de formatos arbitrários (não é necessário assumir que a partição possui um formato específico) (VENTORIM, 2021), a não necessidade de infor- mar como parâmetro o número de grupos no momento da execução e a capacidade de identificar e separar os ruídos (ESTER *et al.*, 1996).

5. Avaliação: A avaliação consiste em aplicar métodos que comprovem a eficácia ou não da solução proposta. Neste caso, foram aplicados os seguintes métodos:

    - Coeficiente da silhueta: Um método para avaliar a qualidade dos *clusters* e avaliar o quão bem os *clusters* estao separados e o quão compactos os *clusters* são. A métrica para se medir a qualidade dos *clusters* gerados (HAN; PEI; KAMBER, 2011).
    - Método do cotovelo: encontrar o número ideal de agrupamentos. Esta tecnica simula diversas divisões em número crescente de grupos e calcula as variâncias internas de cada grupo, buscando o ponto de equilíbrio (LEAL *et al.*, 2019).
    - PCA: método para redução de dimensionalidade, destinado para gerar dados em um número menor de dimensões, facilitando a sua visualizaçãos, ou seja, permite gerar um subconjunto de componentes que pode ser utilizado como uma nova representa- ção (reduzida) dos dados originais (AOUN; NASCIMENTO; SILVA, 2018). Sendo assim, esse método foi utilizado para facilitar a visualização dos possíveis *clusters* gerados, o que se tornou difícil, considerando o número de dimensões dos *embed- dings* de cada texto.
    - TSN-e: método para redução de dimensionalidade, o qual, a partir de um conjunto de pontos em um espaço multidimensional, encontra uma representação fiel desses pontos em um espaço de dimensão menor de forma não-linear e se adaptando

aos dados (ALVES *et al.*, 2019), de forma que apresente como resultado diferentes transformações em diversas regiões desse espaço. O objetivo de utilizar-se desse método foi o mesmo do PCA.

O próximo capítulo irá apresentar os resultados do desenvolvimento da proposta aqui apresentada, bem como as discussões relacionadas.

4 RESULTADOS

Neste capítulo serão mostrados e analisados os resultados obtidos após estudo e implementação da metodologia apresentada. Primeiramente é mostrado um resultado obtido do trabalho, que seria a base em portugues e logo após, é apresentada a proposta de sistema para Help-Desk semiautomático, na qual deve ser aplicada a metologia proposta para esse trabalho. Em seguida, serão apresentados os resultados obtidos ao aplicar o método de agrupamento selecionado à representação textual dos *bugs* de software, no formato de *embeddings* BERT.

4.1 Geração da Base de dados

A Figura 14 representa a base de dados original, em inglês. Como um dos resultados desse trabalho, foi traduzida a base do inglês para o Português conforme a Figura 15, obtendo uma base de dados de *bugs* em Português.

Figura 14 – Bug Reporting BugZilla

ID	Type	Summary	Product	Comp	Assignee▲	Status ▲	Resolution	Updated
1703525		I have unselected CanConfirm option for a group for a particular product. But, the user of that group is able to confirm a bug	Bugzilla	Creating/Changing Bu	create-and-change	UNCO	---	2021-04-07
1600646		localhost/bugz/rest/version giving software error	Bugzilla	WebService	general	UNCO	---	2020-06-25
1678670		undef error - Validation failed for type named NonEmptyStr declared in package Specio::Library::String (/usr/local/share/perl5/Specio/Library/String.pm) at line 46 in sub named (eval) with value undef Trace begun at Specio::Exception->new line 57	Bugzilla	Bugzilla-General	general	UNCO	---	2020-11-30
1439683		Multiple graphical bugs on high refresh rate monitors	Core	Graphics	nobody	UNCO	---	2022-05-20
1452488		Mac: even after bug 1419851 fix, WebRender ON uses more CPU than Web Render OFF for CSS animations	Core	Graphics: WebRender	nobody	UNCO	---	2021-12-08

Fonte: Bugzilla's (2021).

A Figura 15 apresenta alguns itens da base de dados gerada para o trabalho.

Figura 15 – Conjunto de textos

```
ID do bug,produtos,Componente,Cessionário,Status,Resolução,Resumo,Mudado
254085,MDT,Releng,mdt-releng-inbox,FECHADO,FIXO,Novo e notável,06/10/2016 13:04:54
398201,Plataforma,Releng,david_williams,VERIFICADO,FIXO,[CBI] Problemas com org.eclipse.osgi (3.9?) I
190811,z_Archived,TPTP,lizdancy,FECHADO,FIXO,Necessidade de declarar novo probekit API em 4.4 provis
492482,Papiro,Diagrama,dar.a.damus,RESOLVIDO,FIXO,"[Exibição de estereótipo] Desfazer a falha ""Mos
558098,Comunidade,A infraestrutura,webmaster,RESOLVIDO,FIXO,[CRÍTICO] Não foi possível modificar o
206731,z_Archived,TPTP,xubing,FECHADO,FIXO,Falhas do Agent Controller / Falhas de definição de perfil,
102244,WTP Webservices,jst.ws,pmoogk,FECHADO,FIXO,Não é possível criar serviço da Web no projeto d
542552,Comunidade,Servidores,webmaster,FECHADO,FIXO,[Mac OSX] Build machine para SWT nativos pa
461674,Equinócio,Lançador,pascal,RESOLVIDO,FIXO,[Mac] Procure eclipse.ini em um novo lugar,2015-03-
254112,MDT,Releng,Borlander,FECHADO,FIXO,Trabalhar juntos,06/10/2016 13:06:46
229189,z_Archived,TPTP,jgwest,FECHADO,FIXO,Erros aleatórios intermitentes ao lançar um teste contra a
191537,z_Archived,EODM,Izhangl,VERIFICADO,FIXO,As versões do recurso / plug-in precisam ser alterada
492829,Plataforma,SWT,plataforma-swt-inbox,FECHADO,DUPLICADO,"[GTK3] Importar projetos existente
```

Fonte: Autoria própria (2022).

Observa-se nos textos apresentados, que os mesmos contêm muitos ruídos (ex. caracteres especiais, *tags*, etc). Dessa base de dados, foi utilizada apenas a coluna "Resumo"da Fi-

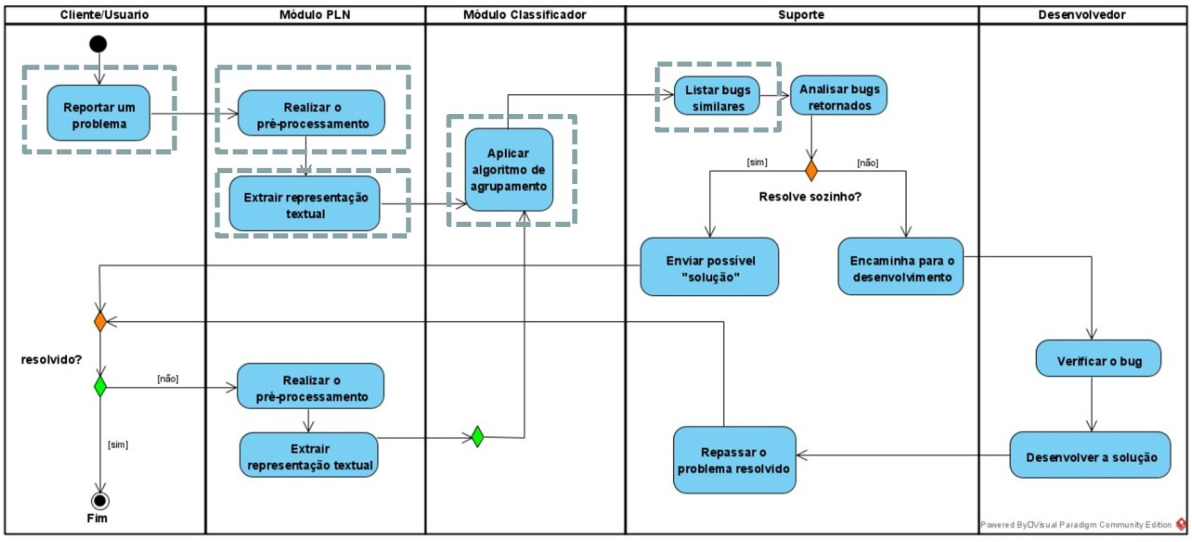
gura 15. A fim de preparar os dados para as próximas etapas, foi realizado o pré-processamento, conforme descrito na subseção 4.4.1.

4.2 Escopo do sistema para *Help-Desk*

O método proposto visa explorar o texto de *bugs* de software, obtidos via solicitações de usuários, objetivando facilitar o trabalho do pessoal do suporte, bem como dos desenvolvedores em sistemas de *help-desk*. A ideia principal é que um sistema que fizer uso desse método possa informar ao suporte sobre *bugs* similares já resolvidos, o que ocorrerá fazendo uso de métodos de PLN e aprendizado de máquina. Desta forma, será apresentado um subconjunto de respos- tas possíveis ao usuário, agilizando assim o atendimento. Além disso, o problema poderá ser encaminhado para o pessoal de desenvolvimento, apenas caso realmente haja necessidade.

A proposta de um futuro sistema usando o método proposto neste trabalho, consiste em atuar como um suporte semiautomático, baseado em IA, em que de um lado estará o usuário informando um problema de software no formato de texto, e do outro estará uma aplicação re- cebendo a solicitação e buscando possíveis respostas ao seu problema. Essa comunicação entre o usuário e o departamento de suporte, ocorrá por meio do repasse de um texto descrevendo o problema encontrado. Esse relato do problema poderá ocorrer via uma interface específica ou mesmo via *chat* ou e-mail. Com base nesse relato, ocorrerá o agrupamento de casos similares. Assim, as respostas mais aproximadas serão apresentadas ao atendente de suporte para que ele possa selecionar a mais adequada, com base na sua experiência, e encaminhá-la ao cliente. Isso fará com que a equipe de suporte e desenvolvimento não gastem tempo desnecessário com pro- blemas já resolvidos anteriormente, ou problemas que podem ser resolvidos mais facilmente. A figura Figura 16 mostra o processo geral do sistema de *help-Desk* proposto.

Figura 16 – Diagrama de atividade - *Help-Desk*



Fonte: Autoria própria (2022).

A forma pontilhada é o método implementado nesse trabalho, e o restante seria a proposta geral do sistema de *help-desk*, que seria a continuação desse trabalho.

Desta forma, esse método objetiva aplicar a representação textual baseada em contexto, conforme as especificações do modelo de linguagem pré-treinado BERT. Em seguida, essas representações serão submetidas a algoritmos de agrupamento, visando agrupar os *bugs* simi- lares, e com isso retornar possíveis soluções com base em analogia. Não é pretendido para o escopo desse trabalho, o desenvolvimento de uma interface gráfica, pois trata-se de um teste de um futuro método a ser aplicado, com foco no *backend*.

4.3 Especificação de Requisitos do Sistema de *Help-Desk*

A seguir é apresentado a especificação dos requisitos funcionais e não-funcionais para o sistema proposto, considerando que a base de dados de *bugs* a ser utilizada não será rotulada. Sendo assim, os quadros abaixo representam os requisitos preliminares para a proposta desse sistema.

Quadro 3 – Requisito 1 - Atender usuário
Fonte: Autoria própria (2022).

F1 Atender Usuário
Descrição: O sistema deve permitir o atendimento de suporte semiautomático para um pro- blema do usuário. Isso deverá ser realizado com base no texto contendo a descrição do problema, o qual será enviado pelo usuário por meio do sistema de *help-desk* ou e-mail.
Requisitos Não-Funcionais:
NF 1.1 Agrupamento de *bugs* similares: Após realizado o pré-processamento do texto de entrada, dado pelo usuário, o sistema deve realizar a busca dos *bugs* mais similar(es), fazendo uso de um método de agrupamento (não-supervisionado).
NF 1.2 Conferência manual: Após realizado o agrupamento dos X *bugs* mais similares (X deverá ser definido), a lista de *bugs* deverá ser analisada por um atendente do suporte, permitindo a ele selecionar a melhor resposta ao usuário, ou ainda encaminhar o *bug* ao departamento responsável.
NF 1.3 Criação de *ticket*: Com os dados retornados pelo módulo de agrupamento, seguido de inferência por parte do suporte, caso um problema não posso ser resolvido pelo suporte, deve-se gerar um *ticket* para o setor de desenvolvimento, o qual deverá conter o máximo de detalhamento possível.
NF 1.4 Contato com o cliente: Após a resolução do *ticket* cadastrado, o suporte deve

Quadro 4 – Requisito 2 - Preparar dados
Fonte: Autoria própria (2022).

F2 Preparar dados
Requisitos Não-Funcionais:
Descrição: São usados métodos de PLN necessários para preparar os dados textuais de *bugs*, os quais servem de entrada no treinamento do modelo de aprendizado, bem como os textos usados como entrada para a busca de *bugs* similares.
Requisitos Não-Funcionais:
NF 2.1 Métodos de pré-processamento: Devem ser aplicados os seguintes métodos de pré-processamento: remoção de caracteres especiais, remoção de *stopwords* e

Quadro 5 – Requisito 3 - Suporte ao cliente
Fonte: Autoria própria (2022).

F3 Suporte ao cliente
Descrição: O suporte deverá intermediar a situação das solicitações dos usuários.
Requisitos Não-Funcionais:
NF 3.1 Entrada de dados: Por meio do relato de problema do usuário, o sistema deve receber, ler o texto e realizar o pré-processamento necessário para que o modelo de inferência possa interpretar os dados de entrada. Feito isso, deverá ser aplicado o método de agrupamento a fim de identificar a que grupo pertence o problema relatado pelo usuário, e quais os *bugs* similares existentes.
NF 3.2 Complementação de dados: O atendente do suporte deverá realizar uma análise dos *bugs* mais similares retornados pelo sistema e, com base na sua experiência, direcionar alternativas de solução ao usuário. Considerando que o atendente classifique o *bug* como complexo e não tenha possibilidade de resolver, esse será encaminhado ao

4.4 Práticas de PLN

Essa seção apresenta os resultados da aplicação de técnicas de pré-processamento, tokenização, e aplicação do *Sentence Transformer* para extração da representação textual a partir do modelo BERT. Em seguida, a representação desses textos foi usada como base de dados de entrada no método de agrupamento DBSCAN. Sendo assim, nessa seção serão apresentadas as etapas da implementação do método proposto para nesse trabalho.

4.4.1 Etapas do Pré-processamento

O pré-processamento consiste em preparar e organizar a base de dados, eliminando as *stopwords*, ou palavras indesejáveis, buscando preparar os textos para serem utilizados no método de representação textual. Não foram aplicadas técnicas de lematização ou *stemming*, pois poderiam dificultar a obtenção do contexto de cada sentença.

Abaixo serão descritas as etapas do pré-processamento realizadas sobre os textos da coluna "Resumo".

1. Remoção de *stopwords*

   Para remoção de *stopwords* foi criada uma função em que as palavras ou *tokens* considerados irrelevantes para a identificação do contexto das sentenças apresentadas (ex. pontuação, digitos, *tags*), foram excluídos do texto. O código correspondente à essa implementação pode ser observado na Figura 17.

   A Figura 18 apresenta os textos originais, antes da retirada das *stopwords* e a Figura 19 o resultado após os processos de remoção de *stopwords*, ambos explicados na seção 2.3. Para a remoção de *stopwords*, primeiramente foi realizada utilizando uma função em que foram informadas palavras que seriam consideradas uma *stopwords*, de acordo com as características do texto (ex. *tags* HTML), e em seguida foi aplicada uma função para

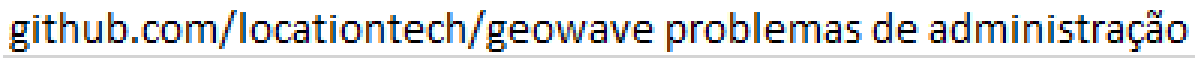
Figura 17 – Remoção de *stopwords* - Pré-processamento - codificação.

```
def CleanText(Text):
    Text = re.sub(r'html',' ',Text)
    Text = re.sub(r'{html}',' ',Text)
    Text = re.sub(r'<div>',' ',Text)
    Text = re.sub(r'<p>',' ',Text)
    Text = re.sub(r'<pre>',' ',Text)
    Text = re.sub(r'<code>',' ',Text)
    Text = re.sub(r'html',' ',Text)
    Text = re.sub(r'< div>',' ',Text)
    Text = re.sub(r'<div>',' ',Text)
    Text = re.sub(r'< p>',' ',Text)
    Text = re.sub(r'<p>',' ',Text)
    Text = re.sub(r'< pre>',' ',Text)
    Text = re.sub(r'< code>',' ',Text)
    Text = re.sub(r'< code>',' ',Text)
    trans_punct = str.maketrans('', '', string.punctuation)
    trans_digit = str.maketrans('', '', string.digits)
    Text = Text.translate(trans_punct)
    Text = Text.translate(trans_digit)
    Text = Text.lower()
    return Text

todasBases['produtos']= todasBases['produtos'].apply(CleanText)
todasBases['Componente']= todasBases['Componente'].apply(CleanText)
todasBases['Cessionário']= todasBases['Cessionário'].apply(CleanText)
todasBases['Status']= todasBases['Status'].apply(CleanText)
todasBases['Resolução']= todasBases['Resolução'].apply(CleanText)
todasBases['Resumo']= todasBases['Resumo'].apply(CleanText)
todasBases['Mudado']= todasBases['Mudado'].apply(CleanText)
```

Fonte: Autoria própria (2022).

remoção de *stopwords* própria da biblioteca *spacy*.

Figura 18 – Antes da retirada das *stopwords* e tokenização



Fonte: Autoria própria (2022).

Figura 19 – Após a retirada das *stopwords* e tokenização

Fonte: Autoria própria (2022).

2. Tokenização

A Figura 20 apresenta a aplicação da tokenização, a qual consiste em dividir um texto em entidades menores, ou seja, em pegar o texto original completo e separar cada uma das sentenças em tokens. A tokenização foi necessária, pois é um procedimento fundamental, tanto para o pré-processamento quanto para para efetuar uso do BERT, que obrigatoria- mente precisa que os textos estejam ’tokenizados’.

Figura 20 – Tokenização - codificação

```
import pandas as pd
from spacy.lang.pt.stop_words import STOP_WORDS
stopwords = STOP_WORDS
!pip install deplacy
!python -m spacy download pt_core_news_sm
import pkg_resources,imp
imp.reload(pkg_resources)
import spacy
pontuacoes ='!"#$%¨&*()`;[]{}=_.+-*/~,.'
from google.colab import drive
drive.mount('/content/drive')
nlp=spacy.load("pt_core_news_sm")

todasBases = pd.read_csv('/content/drive/MyDrive/6Periodo/TCC2/NovoMetodo/BasePreProcessada/testeBaseNova.csv', encoding='utf-8')

def preprocessamento(texto):
  texto = texto.lower()
  documento = nlp(texto)
  lista = []
  for token in documento:
    lista.append(token.lemma)#lematização
  lista = [palavra for palavra in lista if palavra not in stopwords and palavra not in pontuacoes]
  #remoção de stopwords e pontuacoes
  return lista

todasBases[['produtos','Componente', 'Cessionario', 'Status','Resolucao', 'Resumo']]
= todasBases[['produtos','Componente', 'Cessionario', 'Status','Resolucao', 'Resumo']].astype(str)

todasBases['produtos']=todasBases['produtos'].apply(preprocessamento)
todasBases['Componente'] = todasBases['Componente'].apply(preprocessamento)
todasBases['Cessionario'] = todasBases['Cessionario'].apply(preprocessamento)
todasBases['Status'] = todasBases['Status'].apply(preprocessamento)
todasBases['Resolucao'] = todasBases['Resolucao'].apply(preprocessamento)
todasBases['Resumo'] = todasBases['Resumo'].apply(preprocessamento)

todasBases.to_csv('baseNova.csv')
```

Fonte: Autoria própria (2022).

4.4.2 Extração da Representação Textual

A etapa de pré-processamento foi necessária para preparar os dados textuais para serem submetidos ao *framework Sentence_Transformer* (REIMERS, 2020), utilizado para gerar os *embeddings* do BERT. A Figura 21 mostra a listagem do código Python utilizado para a geração dos *embeddings* BERT, sendo seu resultado mostrado na Figura 22. Foi utilizado o método *cosine-similarity* passando como parâmetro a partir do vetor 0, filtrando pela coluna Resumo da base, e com o modelo pré-treinado passado como parâmetro no *Sentence_Transfor- mer*, mapeando frases e parágrafos para um espaço vetorial denso de 768 dimensões e pode ser usado para tarefas como agrupamento. E o *sentence embeddings* recebe esses dados, transfor- mando o modelo de sentença da coluna Resumo, para posteriormente utilizar no método *cosine similarity* (semelhança de cosseno). Esse método foi escolhido pois é uma medida muito co- nhecida na área de PLN, pois tem seus resultados potencializados, com a capacidade de, nos seus resultados (OLIVEIRA *et al.*, 2019), ele mede a similaridade entre listas de vetores calcu- lando o ângulo de cosseno entre as duas listas de vetores, para então encontrar frases com um significado semelhante.

O método *Sentence_Transformer* tem uma facilidade, pois procura o modelo pré-treinado conforme passado como parâmetro. Nesse trabalho foi aplicado o *multidistilbert-base- uncased*, que é um modelo multilíngue BERT, treinado em texto com todas as letras minúsculas. A aplicação acontece automaticamente no momento em que executar o código, e outro destaque é que não precisa mais gerar os *embeddings* para cada sentença, a partir dos *embeddings* de cada token, como era feito a pouco tempo atrás, pois o *Sentence_Transformer* facilita esse trabalho, retornando o *embedding* para a sentença.

Figura 21 – BERT - codificação

```
import pandas as pd
import numpy as np
!pip install -U sentence-transformers
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
from google.colab import drive

drive.mount('/content/drive')

base = pd.read_csv('/content/drive/MyDrive/6Periodo/TCC2/NovoMetodo/BaseTokenizada/baseNova.csv', encoding='utf-8')
data_label = base.Resumo.values

model = SentenceTransformer('abhijithneilabraham/stsb_multi_mt_distilbert-base-uncased')

sentence_embeddings = model.encode(data_label)

sentence_embeddings.shape

cosine_similarity(
    [sentence_embeddings[0]],
    sentence_embeddings[1:]
)

len(sentence_embeddings)
np.savetxt('BertBase.csv', sentence_embeddings, delimiter=",")
```

Fonte: Autoria própria (2022).

A Figura 22 mostra apenas algumas das 768 dimensões que compõe cada *embedding* gerado pelo BERT.

Figura 22 – *Embeddings* BERT correspondentes a cada texto.

```
[[ 0.19881484 -0.16296445  0.35760546 -0.72817177 -0.39896196  0.22920296
   0.02650207  0.6077618  -0.7088234   0.13595846 -0.30558532 -0.21168676
   0.188298   -0.17722552  0.21753238  0.65540445  0.3516778  -0.1503949
  -0.10421658  0.02088168  0.67095697 -0.34441057  0.04399243 -0.40829232
  -0.05833855  0.36822003  0.15426788 -0.3905072  -0.13283192 -0.8305995
   0.06373966 -0.27689663  0.05993375  0.22478403  0.44897267 -0.17491655
  -0.2021784  -0.34636286 -0.36507693  0.31167695 -0.8554128  -0.04618547
  -0.13649723  0.442452    0.09492387  0.38109517  0.10300308  0.42206946
  -0.54395276  0.37001923 -1.203664   -0.31230348  1.2800772   0.17484255
   0.15851416  0.29983735 -0.00714554 -0.29179984  0.5974175  -0.781544
   0.06504731 -0.48687005 -0.1397386   0.14600226 -0.28921103 -0.50453025
  -0.11783384  0.3353093   0.18726745 -0.20822623 -0.45161623  0.1284636
  -0.5334988   0.14206798  0.22033188 -0.08780081  0.5768315  -0.21538131
   0.65658706 -0.14042625 -0.8934711   0.11610802  0.34180957 -0.11269136
   0.5842282  -0.43872288 -0.30522826  0.12407978 -0.10084499  0.644547
```

Fonte: Autoria própria (2022).

4.4.3 Aplicando o Algoritmo de Agrupamento - DBSCAN

O método de agrupamento DBSCAN foi escolhido para uso nesse trabalho por ser um método de agrupamento não-supervisionado, o que nesse trabalho se encaixaria bem para clus- terizar os *bugs* não rotulados da base de textos criada. Além disso, segundo Daniel (2016), o DBSCAN tem um diferencial, não há necessidade de ter um conhecimento preliminar dos dados analisados.

A Figura 23 mostra o código utilizado na implementação do DBSCAN, no qual é possível observar os principais hiperparâmetros utilizados conforme Quadro 6.

O método DBSCAN teve como objetivo agrupar os dados, fazendo a previsão do agrupamento com base nos seus hiperparâmetros. Os principais hiperparâmetros utilizados foram:

Quadro 6 – Hiperparâmetros utilizados no DBSCAN.

Experimento	eps	minSamples	nSamples
Exp 1	0.5	0.5	40.000
Exp 2	4	1536	10.000
Exp 3	0.5	1536	40.000

Fonte: Autoria própria (2022).

No Exp 1 O valor de *eps* = 0.5 foi utilizado por ser padrão do método DBSCAN. Desta forma, esse valor foi aplicado com o hiperparâmetro *minSamples* (na literatura também encon- trado como *min_Points*) igual a 0.5, padrão do método). No Exp2 e Exp 3, o valor de *min_Points* foi igual a 1536, valor esse obtido de acordo com (MULLIN, 2020), que afirma que o valor ideal

Figura 23 – DBSCAN - codificação

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
from google.colab import drive
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

drive.mount('/content/drive')
base = pd.read_csv('/content/drive/MyDrive/6Periodo/TCC2/NovoMetodo/BERT/BertTokenizada.csv', engine='python', encoding='utf-8')
base, labels_true = make_blobs(n_samples=40000, cluster_std=4, random_state=0)
base = StandardScaler().fit_transform(base)
db = DBSCAN(eps=4, min_samples=1536).fit(base)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1]
    class_member_mask = labels == k
    xy = base[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0],xy[:, 1],"o",markerfacecolor=tuple(col),markeredgecolor="k",markersize=14,)
    xy = base[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0],xy[:, 1],"o",markerfacecolor=tuple(col),markeredgecolor="k",markersize=6,)

plt.title("Número estimado de clusters: %d" % n_clusters_)
plt.show()
```

Fonte: Autoria própria (2022).

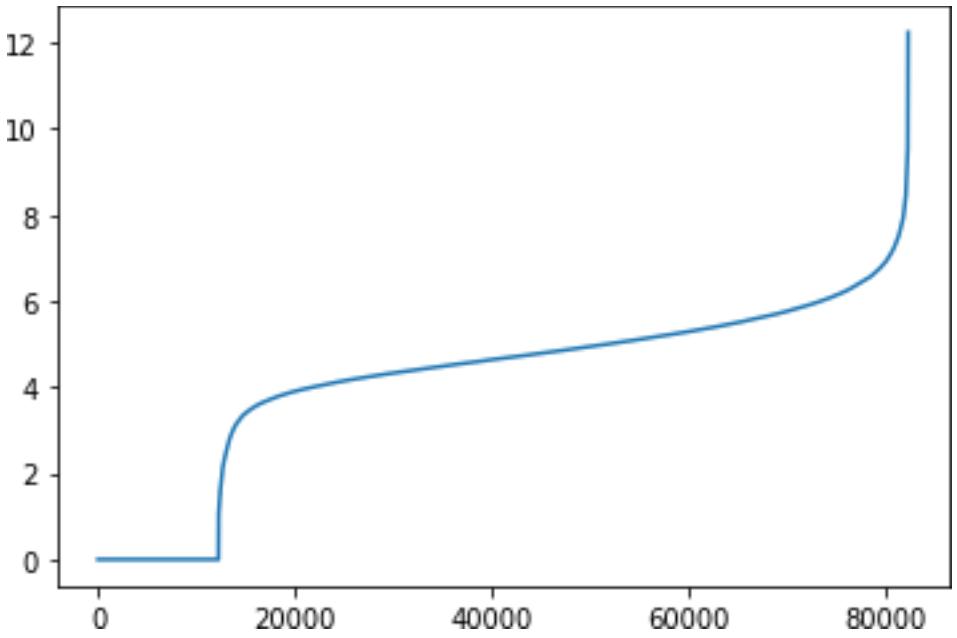
para *minSamples* seria de duas vezes a dimensão da base de dados. Ainda com o valor de *eps* = 0.5 (Exp 1 e Exp 3), foi aplicado o *nSamples* igual a 40.000 (valor máximo suportado pelo DBSCAN em função de seu custo computacional se tornar muito elevado). Destaca-se que essa limitação do método de agrupamento em permitir um valor de *nSamples* maior e mais apropri- ado, conforme sugere o método do cotovelo Figura 24 pode ter comprometido o resultado do método proposto. Já no Exp 2, o valor de eps = 4 foi sugerido pelo método do cotovelo e foi utilizado com *minSamples/min_Points* igual a 1536.

Mesmo alterando os hiperparâmetros, os resultados se mantiveram, o que revela que não há uma estrutura subjacente nessa base de dados. Pelo menos não foi revelada usando apenas o DBSCAN. Sendo assim, buscando melhorar os resultados ou comprovar os resultados obtidos com o DBSCAN, foram utilizados dois algoritmos de redução de dimensionalidade, conforme relatados na subseção 4.4.4.

4.4.4 Redução de dimensionalidade

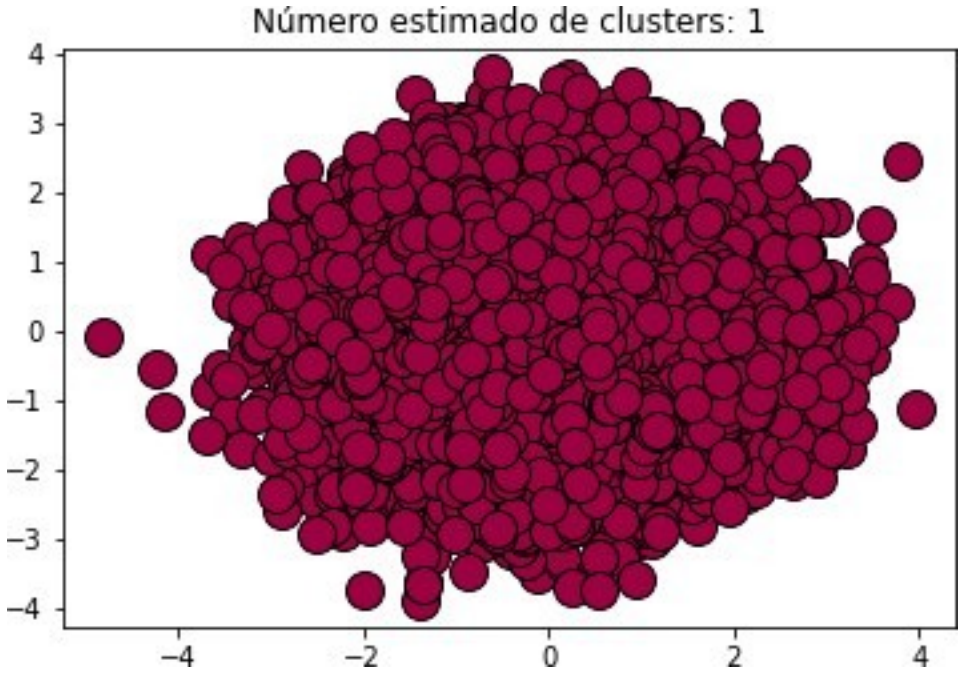
Foram aplicados dois métodos populares de redução de dimensionalidade: (PCA e TSN- e). Ambos servem para projetar uma dimensão menor dos dados, mantendo-os consistentes. Como a base possui 768 dimensões (alta dimensionalidade), o uso da redução para visualizá-los

Figura 24 – Método do cotovelo.



Fonte: Autoria própria (2022).

Figura 25 – Agrupamento via DBSCAN.



Fonte: Autoria própria (2022).

seria de suma importância. O objetivo da PCA é converter um conjunto de variáveis altamente correlacionadas a um conjunto de variáveis independentes, usando transformações lineares para conseguir uma redução de variáveis. Esse método é recomendável quando há um grande nú- mero de variáveis numéricas dificultando a sua visualização (VALENZUELA; SCHWARTZ; PEDRINI, 2011).

Figura 26 – PCA - codificação.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from google.colab import drive
import numpy as np
from sklearn.decomposition import PCA
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/6Periodo/TCC2/NovoMetodo/BERT/BertTokenizada.csv', encoding='utf-8')
rng = np.random.RandomState(1)
X = data
pca = PCA(n_components=2)
pca.fit(data)
X_pca = pca.transform(data)
X_new = pca.inverse_transform(X_pca)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```

Fonte: Autoria própria (2022).

O PCA da Figura 26 mostra os dados transformados que foram reduzidos a 2 (duas) dimensões, através do parâmetro *n_Components*. Para entender o efeito dessa redução de dimensionalidade, pode-se notar que os pontos da Figura 27 são apresentados em duas dimensões, o que facilitou a interpretação do resultado, mostrando que realmente não há *clusters* bem definidos.

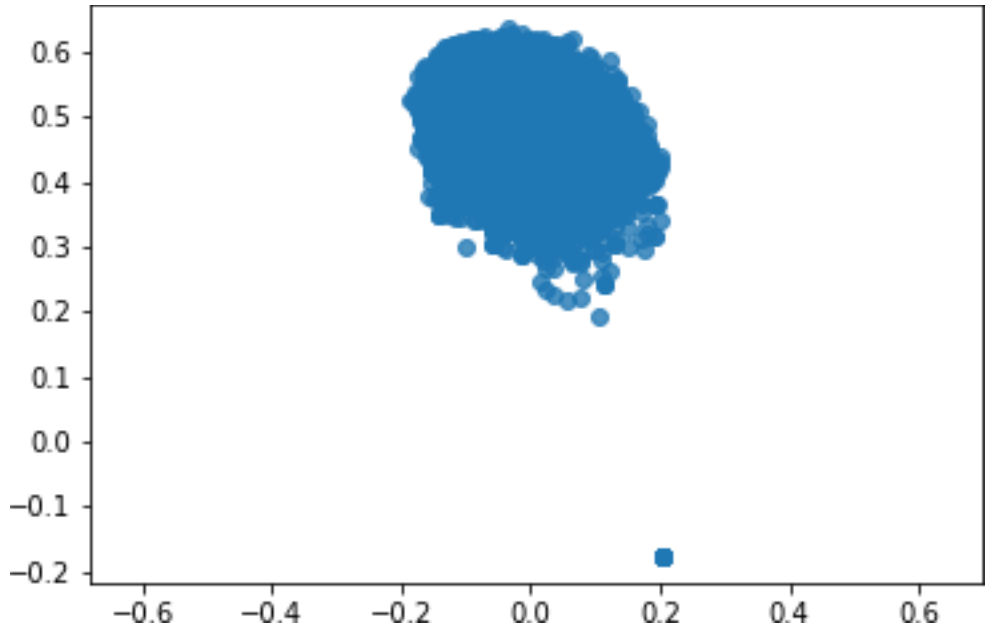
Sendo assim, a hipótese de que a base de dados utilizada não possui uma estrutura clara para dados representados ao aplicar o DBSCAN, foi reforçada com os resultados obtidos com o PCA, o que revelou que as frases vistas na base não mostram similaridade em seu contexto, ou seja, não justifica estarem no mesmo agrupamento.

Ao aplicar a redução de dimensionalidade a partir do método TSN-e (Figura 28), o resultado apresentado é o da Figura 29. Para entender esse resultado, um resultado obtido com a redução de dimensionalidade do TSN-e que ao verificar na base, nos retorna as frases da base.

Os três algoritmos testados (PCA, TSN-e e DBSCAN), obtiveram resultados similares (onde o DBSCAN apresentou somente um *cluster*, o PCA revelou somente um *cluster*, e o TSN-e revelou dois *clusters* e alguns ruídos (Figura 30).

Desta forma, ao avaliar amostras de textos obtidos de pontos pertencentes ao *cluster* 1 (Figura 30) e de pontos pertencentes ao *cluster* 2 (Figura 31), não foi observado padrão semân- tico entre os textos retornados. Assim, tem-se mais uma constatação de que a base não possui uma subjacência nos dados, o que ficou claro em todos os resultados obtidos e demonstrados no presente trabalho.

Figura 27 – Gráfico do PCA.



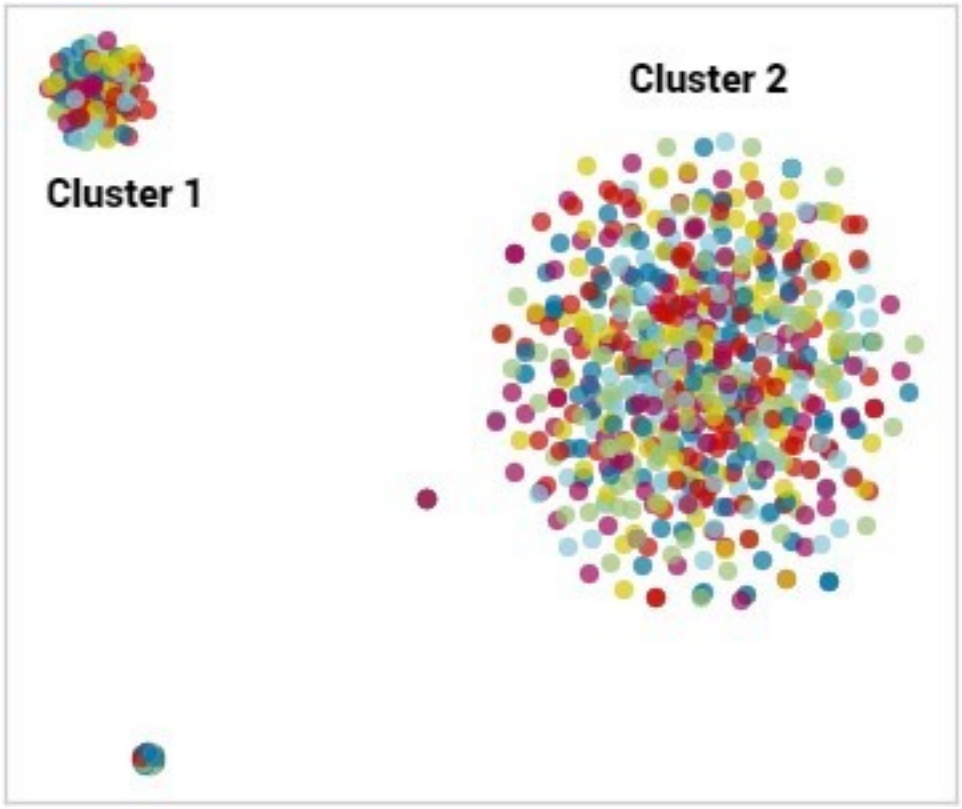
Fonte: Autoria própria (2022).

Figura 28 – TSN-e - codificação.

```
from google.colab import drive
import pandas as pd
import numpy as np
drive.mount('/content/drive')

df = pd.read_csv('/content/drive/MyDrive/6Periodo/TCC2/NovoMetodo/BERT/BertBase.csv', encoding='utf-8')

from sklearn.feature_extraction.text import TfidfVectorizer
from yellowbrick.text import TSNEVisualizer
from yellowbrick.datasets import load_hobbies

tfidf = TfidfVectorizer()

X = tfidf.fit_transform(corpus)
y = corpus.columns


tsne = TSNEVisualizer()

tsne.fit(X, y)

tsne.show()
```

Fonte: Autoria própria (2022).

Figura 29 – Resultado gráfico do TSN-e.



Fonte: Autoria própria (2022).

Figura 30 – Texto correspondente ao *cluster* 1

erro de atualização com
amostra ecore representar referência com a referência eopposite especificada como um único link de diagrama
o pacote do table modelo de implantação api
crie um vídeo de introdução edt hello world

Fonte: Autoria própria (2022).

Figura 31 – Texto correspondente ao *cluster* 2

crie um vídeo de introdução edt hello world
tema melhora a usabilidade do foco
anotação idclass incorretamente necessária anotação id presente usando tabela por locatário multilocação
alterar o valor constante não causa recompiação do código que o usa

Fonte: Autoria própria (2022).

5 CONCLUSÃO

O objetivo geral deste trabalho foi propor uma metodologia semiautomática para agrupar *bugs* de software em Português com base em analogia, aplicando PLN associado a métodos de aprendizado de máquina. Para isso, inicialmente foi gerada uma base de dados de *bugs* de software na língua portuguesa, o que não existia até o momento, sendo esse um dos resultados desse trabalho.

Em seguida foram aplicados métodos de pré-processamento sobre a base de dados, para então extrair a representação textual, que foi usada no método de agrupamento. Ao submeter a representação textual ao método DBSCAN e métodos de redução de dimensionalidade (PCA e TSN-e), não foram obtidos *clusters* bem definidos ou significativos com relação aos dados representados. Em outras palavras, não foi obtida uma estrutura subjacente da base de dados utilizada.

Com base nos trabalhos existentes para a classificação de textos com o objetivo de dar uma resposta à um usuário de forma ágil em sistemas de *help-desk*, o uso de métodos de PLN e de aprendizado de máquina aplicados para uma base textual de *bugs* de software, teria a possibilidade e o potencial de agilizar os atendimentos de usuários/clientes na solução de seus problemas.

Um ponto positivo desse trabalho é que as tecnologias utilizadas não são aprendidas no decorrer do curso. Então, o desenvolvimento desse trabalho proporcionou um conhecimento diferenciado ao acadêmico sobre BERT, PLN e aprendizado de máquina, o que apesar de ser um desafio, foi levado com motivação e muita persistência.

Apesar do resultado obtido não ter sido positivo com relação ao objetivo da metodologia proposta (agrupamento de *bugs*), foi possível chegar a algumas hipóteses e sugestões para trabalhos futuros, visando a melhoria dos resultados, tais como:

- Realizar testes com outros métodos de agrupamento que suportem um número maior de amostras.
- Aplicar outros modelos pré-treinados similares ao BERT, como por exemplo os modelos da família GPT-2 e GPT-3.
- Complementação da base de dados contendo textos de *bugs* de software em Português. Apesar de ter sido um desafio, foi gerada uma base de textos e traduzí-la para o Português brasileiro. Mas para validar os resultados adequadamente, seria importante uma base mais volumosa.

REFERÊNCIAS

ABUSHAWAR, Bayan; ATWELL, Eric. Alice chatbot: Trials and outputs. **Computación y Sistemas**, Centro de Investigación en Computación, IPN, v. 19, n. 4, p. 625–632, 2015. Citado na página 22.

AKBIK, Alan; BERGMANN, Tanja; VOLLGRAF, Roland. Pooled contextualized embeddings for named entity recognition. In: **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**. [S.l.: s.n.], 2019. p. 724–728. Citado na página 25.

ALEEM, Saiqa; CAPRETZ, Luiz Fernando; AHMED, Faheem. Benchmarking machine learning technologies for software defect detection. **arXiv preprint arXiv:1506.07563**, 2015. Citado na página 26.

ALVARENGA, João Paulo Reis. Avaliação de métodos de transferência de aprendizado aplicados a problemas de processamento de linguagem natural em textos da língua portuguesa. 2019. Citado na página 24.

ALVES, Ivyna Rayany Santino *et al.* Análise automática do posicionamento de parlamentares sobre a reforma da previdência por meio de seus discursos. Universidade Federal de Campina Grande, 2019. Citado na página 42.

ANCHIÊTA, Rafael *et al.* Pln: Das técnicas tradicionais aos modelos de deep learning. **Sociedade Brasileira de Computação**, 2021. Citado na página 37.

AOUN, Paulo Henrique Calado; NASCIMENTO, Andre CA; SILVA, Adenilton J da. Evaluation of dimensionality reduction and truncation techniques for word embeddings. In: SBC. **Anais do XV Encontro Nacional de Inteligência Artificial e Computacional**. [S.l.], 2018. p. 903–911. Citado na página 41.

BANDEIRA, Andre Luis Martins *et al.* **Classificação automática da prioridade de defeitos utilizando seleção de atributos: um estudo de caso na indústria**. 2019. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2019. Citado na página 15.

BARBOSA, Eiji Adachi Medeiros. **Código de Tratamento de Exceções**. 2012. Tese (Doutorado) — PUC-Rio, 2012. Citado 2 vezes nas páginas 14 e 19.

BEHZADI, Fateme. Natural language processing and machine learning: A review. **International Journal of Computer Science and Information Security**, LJS Publishing, v. 13, n. 9, p. 101, 2015. Citado na página 15.

BELTRAN, Rafael Duarte. Detecção de fraudes bancárias utilizando métodos de clustering. Universidade Federal do Pampa, 2019. Citado na página 29.

BINDRA, Kamalpreet; MISHRA, Anuranjan. A detailed study of clustering algorithms. In: IEEE. **2017 6th International Conference on Reliability, Infocom Technologies and**

Optimization (Trends and Future Directions)(ICRITO). [S.l.], 2017. p. 371–376. Citado na página 29.

BISHOP, Christopher M. Pattern recognition. **Machine learning**, v. 128, n. 9, 2006. Citado na página 28.

BUGZILLA'S, UI. Bugzilla. 2021. Citado na página 43.

CADIMA, Jorge; JOLLIFFE, Ian. On relationships between uncentred and column-centred principal component analysis. **Pakistan Journal of Statistics**, Citeseer, v. 25, n. 4, 2009. Citado na página 33.

CAMILO, Cássio Oliveira; SILVA, João Carlos da. Mineração de dados: Conceitos, tarefas, métodos e ferramentas. **Universidade Federal de Goiás (UFC)**, v. 1, n. 1, p. 1–29, 2009. Citado na página 28.

CASSIANO, Keila Mara; PESSANHA, José Francisco Moreira. Análise espectral singular com clusterização baseada em densidade na modelagem de séries temporais. 2014. Citado na página 30.

CHAVES, Letícia Saraiva. Utilizando um modelo transformer no processo de identificação de entidades nomeadas em textos criminais. 2021. Citado 2 vezes nas páginas 26 e 27.

CHOWDHURY, Gobinda G. Natural language processing. **Annual review of information science and technology**, Wiley Online Library, v. 37, n. 1, p. 51–89, 2003. Citado na página 21.

DANIEL, Guilherme Priólli. Otimização de algoritmos de agrupamento espacial baseado em densidade aplicados em grandes conjuntos de dados. Universidade Estadual Paulista (UNESP), 2016. Citado 2 vezes nas páginas 38 e 50.

DEVLIN, Jacob *et al.* Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018. Citado 4 vezes nas páginas 16, 26, 36 e 38.

DUDA, Richard O; HART, Peter E; STORK, David G. Unsupervised learning and clustering. **Pattern classification**, Wiley New York, p. 517–601, 2001. Citado na página 28.

DUMAS, Bruno; LALANNE, Denis; OVIATT, Sharon. Multimodal interfaces: A survey of principles, models and frameworks. In: **Human machine interaction**. [S.l.]: Springer, 2009. p. 3–26. Citado na página 15.

ELHADAD, Michael. Natural language processing with python steven bird, ewan klein, and edward loper (university of melbourne, university of edinburgh, and bbn technologies) sebastopol, ca: O'reilly media, 2009, xx+ 482 pp; paperbound, isbn 978-0-596-51649-9, textdollar44. 99; on-line free of charge at nltk. org/book. **Comput. Linguistics**, v. 36, n. 4, p. 767–771, 2010. Citado na página 21.

ELLEN, Brito Moura Gera; MOURA, Raimundo Santos. A semiautomatic model for classification of it services. In: IEEE. **2016 35th International Conference of the Chilean Computer Science Society (SCCC)**. [S.l.], 2016. p. 1–8. Citado na página 34.

ESTER. Simoudis, evangelos; han, jiawei; fayyad, usama m.(eds). a density-based algorithm for discovering clusters in large spatial databases with noise. In: **Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press**. [S.l.: s.n.], 1996. p. 226–231. Citado na página 31.

ESTER, Martin *et al.* A density-based algorithm for discovering clusters in large spatial databases with noise. In: **kdd**. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231. Citado 2 vezes nas páginas 29 e 41.

FILHO, José Evaldo Gonçalves Lopes. O uso de machine learning (t-sne) como técnica para visualização de padrões no programa bolsa família com base nos dados do cadastro único. Universidade Federal de Minas Gerais, 2019. Citado na página 34.

FILHO, Rubem Melendez. **Service Desk Corporativo: Solução com base na ITIL* V3**. [S.l.]: Novatec Editora, 2017. v. 3. Citado 2 vezes nas páginas 19 e 20.

FORMAN, George; KIRSHENBAUM, Evan. Extremely fast text feature extraction for classification and indexing. In: **Proceedings of the 17th ACM conference on Information and knowledge management**. [S.l.: s.n.], 2008. p. 1221–1230. Citado na página 33.

FORMAN, George *et al.* An extensive empirical study of feature selection metrics for text classification. **J. Mach. Learn. Res.**, v. 3, n. Mar, p. 1289–1305, 2003. Citado na página 33.

FURTADO, Maria Inês Vasconcellos. Inteligência competitiva para o ensino superior privado: Uma abordagem através da mineração de textos. **Rio de Janeiro, RJ: UFRJ**, 2004. Citado na página 20.

GANT, Vanya; RODWAY, Susan; WYATT, Jeremy. Artificial neural networks: practical considerations for clinical applications. **Clinical applications of artificial neural networks**, Cambridge University Press Cambridge, p. 329–356, 2001. Citado na página 26.

GAVA, Éverton Marangoni *et al.* O algoritmo density-based spatial clustering of applications with noise (dbscan) na clusterização dos indicadores de dados ambientais. **Anais SULCOMP**,
v. 6, 2013. Citado na página 30.

GHAREHCHOPOGH, Farhad Soleimanian; KHALIFELU, Zeinab Abbasi. Analysis and evaluation of unstructured data: text mining versus natural language processing. In: IEEE. **2011 5th International Conference on Application of Information and Communication Technologies (AICT)**. [S.l.], 2011. p. 1–4. Citado na página 15.

GOLDBERG, Yoav. Neural network methods for natural language processing. **Synthesis lectures on human language technologies**, Morgan & Claypool Publishers, v. 10, n. 1, p. 1–309, 2017. Citado na página 21.

GOMES, Helder Joaquim Carvalheira. **Text Mining: análise de sentimentos na classificação de notícias**. 2013. Tese (Doutorado) — UNL-Universidade Nova de Lisboa, 2013. Citado na página 15.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. Data mining: Concepts and techniques third edition [m]. **The Morgan Kaufmann Series in Data Management Systems**, v. 5, n. 4, p. 83–124, 2011. Citado na página 23.

HAN, Jiawei; PEI, Jian; KAMBER, Micheline. **Data mining: concepts and techniques**. [S.l.]: Elsevier, 2011. Citado na página 41.

HARDENIYA, Nitin *et al.* **Natural language processing: python and NLTK**. [S.l.]: Packt Publishing Ltd, 2016. Citado na página 24.

IEEE. **IEEE**. 2021. Site IEEE. Disponível em: https://www.ieee.org/. Citado na página 19.

LARY, David J *et al.* Machine learning in geosciences and remote sensing. **Geoscience Frontiers**, Elsevier, v. 7, n. 1, p. 3–10, 2016. Citado na página 26.

LEAL, Sidney Evaldo *et al.* Métodos de clusterizaç ao para a criaç ao de corpus para rastreamento ocular durante a leitura de parágrafos em português. 2019. Citado na página 41.

LEITÃO, Wellyngton Amaral. Estudo exploratório de bugs de tratamento de exceção na plataforma android: estudo exploratório. 2016. Citado 2 vezes nas páginas 14 e 19.

LEVY, Omer; GOLDBERG, Yoav. Dependency-based word embeddings. In: **Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)**. [S.l.: s.n.], 2014. p. 302–308. Citado na página 25.

LIMA, William Carneiro. **Estudo de classificadores e construção de dataset para diagnóstico de dengue, chikungunya e zika**. 2017. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2017. Citado na página 23.

LINKE, Leonardo Lavalhos. Implantação de um sistema de help-desk: um estudo de caso na exatus soluções estratégicas. Universidade Federal de Santa Maria, 2015. Citado na página 14.

MADHURI, CH Raga; ANURADHA, G; PUJITHA, M Vani. House price prediction using regression techniques: a comparative study. In: IEEE. **2019 International Conference on Smart Structures and Systems (ICSSS)**. [S.l.], 2019. p. 1–5. Citado na página 29.

MANHÃES, Marcelo Mota. **Classificação e Resolução de Defeitos em Manutenção de Software utilizando ODC e Histórico de Soluções**. 2014. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2014. Citado na página 35.

MARSLAND, Stephen. **Machine learning: an algorithmic perspective**. [S.l.]: Chapman and Hall/CRC, 2011. Citado 2 vezes nas páginas 28 e 29.

MELO, MARCELO DAMASCENO DE. Um processo de mineração de dados para predição de níveis criminais de áreas geográficas urbanas. 2010. Citado na página 40.

MIKOLOV, Tomas *et al.* Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013. Citado na página 25.

MITCHELL, Tom M. **The Discipline of Machine Learning. July 2006**. [S.l.]: CMU-ML-06-108, 2009. Citado na página 27.

MULLIN, Tara. Dbscan parameter estimation using python. **Medium**, p. 1–2, 2020. Citado na página 50.

MUNS, Ron. **The Help Desk Handbook: The Help Desk Institute Guide to Help Desk Operations and Problem Management**. [S.l.]: Help Desk Institute, 1993. Citado na página 20.

NETO, Antônio Cavalcante Araújo. G2p-dbscan: estratégia de particionamento de dados e de processamento distribuído fazer dbscan com mapreduce. 2016. Citado na página 30.

NETO, João Mendes de Oliveira; TONIN, Sávio Duarte; PRIETCH, Soraia Silva. Processamento de linguagem natural e suas aplicações computacionais. **Acesso em**, v. 12, 2010. Citado na página 21.

NORVIG, P; RUSSELL, S. **Inteligência artificial: Traduç ao da 3a Ediç ao**. [S.l.]: Sao Paulo: Elsevier Brasil, 2013. Citado na página 29.

OLIVEIRA, Douglas Camilo de *et al.* Aplicação das técnicas de processamento de linguagem natural cosine similarity e word mover's distance na automatização da correção de questões discursivas no sistema tutor inteligente mazk. Araranguá, SC, 2019. Citado na página 49.

OLIVEIRA, Júlio César Alcântara. Atendimento remoto: estudo de caso sobre o atendimento remoto aos clientes de uma empresa de automação. **Tecnologia em Gestão da Tecnologia da Informação-Unisul Virtual**, 2019. Citado na página 20.

OMAR, Salima; NGADI, Asri; JEBUR, Hamid H. Machine learning techniques for anomaly detection: an overview. **International Journal of Computer Applications**, Foundation of Computer Science, v. 79, n. 2, 2013. Citado na página 26.

PACHECO, Daniel Nunes. Abordagem dos algoritmos de agrupamento k-means e fuzzy c-means na identificação de zonas pluviométricas em santa catarina utilizando o modelo de processo crisp-dm. 2018. Citado na página 29.

PATIL, Tina R. Msss performance analysis of naive bayes and j48 classification algorithm for data classification. intl. **Journal of Computer Science and Applications**, v. 6, n. 2, 2013.
Citado na página 34.

PEREIRA, Mayana; CHRISTIANSEN, Scott. Identifying security bug reports based solely on report titles and noisy data (identificar relatórios de bug de segurança com base apenas em títulos de relatório e dados com ruídos). 2020. Citado na página 15.

PLATT, John *et al.* Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. **Advances in large margin classifiers**, Cambridge, MA, v. 10,
n. 3, p. 61–74, 1999. Citado na página 35.

PUPPO, Elza Meira. **Sistema de recomendação de instituição de ensino superior para atenuar evasão utilizando análise de perfis de egressos**. 2021. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2021. Citado na página 29.

RASCHKA, Sebastian. **Python machine learning**. [S.l.]: Packt publishing ltd, 2015. Citado na página 25.

RÄTSCH, Gunnar. A brief introduction into machine learning. **Friedrich Miescher Laboratory of the Max Planck Society**, Citeseer, 2004. Citado na página 27.

RAZERA, Daniel Espanhol; MACIEL, Carlos Dias. Sistema de classificação nebuloso usando pca robusta. 2010. Citado na página 33.

REIMERS, Iryna Gurevych Nils. Making monolingual sentence embeddings multilingual using knowledge distillation. **arXiv preprint arXiv:2004.09813**, 2020. Citado na página 49.

REIMERS, Nils; GUREVYCH, Iryna. Sentence-bert: Sentence embeddings using siamese bert-networks. In: **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing**. Association for Computational Linguistics, 2019. Disponível em: https://arxiv.org/abs/1908.10084. Citado na página 36.

RIVEROS, Lilian Jeannette Meyer; CAMPOS, Wagner; PERAZZOLLI, Paulo Roberto. Itil aplicada ao service desck–um estudo de caso com a ferramenta solution manager da sap. **Anuário Pesquisa e Extensão Unoesc Videira**, v. 2, p. e15174–e15174, 2017. Citado na página 17.

RODRÍGUEZ, Marcia Marina; BEZERRA, Byron Leite Dantas. Processamento de linguagem natural para reconhecimento de entidades nomeadas em textos jurídicos de atos administrativos (portarias). **Revista de Engenharia e Pesquisa Aplicada**, v. 5, n. 1, p. 67–77, 2020. Citado na página 22.

ROSA, João Luís Garcia. Fundamentos da inteligência artificial. **Rio de Janeiro: LTC**, v. 1, 2011. Citado na página 21.

RUSSEL, Stuart; NORVIG, Peter. Inteligência artificial. tradução da terceira edição. **Rio de Janeiro, RJ: Elsevier Editora**, 2013. Citado na página 17.

SANTOS, Larissa Sayuri Futino Castro dos. Estudo online da dinâmica espaço-temporal de crimes através de dados da rede social twitter. Universidade Federal de Minas Gerais, 2015. Citado na página 26.

SAS. **SAS**. 2021. Site SAS. Disponível em: https://www.sas.com/pt_br/home.html. Citado na página 28.

SCHMITT, Vinícius Fernandes. Uma análise comparativa de técnicas de aprendizagem de máquina para prever a popularidade de postagens no facebook. 2013. Citado na página 28.

SCHNEIDER, Karl-Michael. On word frequency information and negative evidence in naive bayes text classification. In: SPRINGER. **International Conference on Natural Language Processing (in Spain)**. [S.l.], 2004. p. 474–485. Citado na página 33.

SCHUBERT, Erich; HESS, Sibylle; MORIK, Katharina. The relationship of dbscan to matrix factorization and spectral clustering. In: **LWDA**. [S.l.: s.n.], 2018. Citado na página 31.

SCHUBERT, Erich *et al.* Dbscan revisited, revisited: why and how you should (still) use dbscan. **ACM Transactions on Database Systems (TODS)**, ACM New York, NY, USA, v. 42,
n. 3, p. 1–21, 2017. Citado 2 vezes nas páginas 31 e 32.

SERRAS, Felipe Ribas. **Algoritmos baseados em atenção neural para a automação da classicação multirrótulo de acórdãos jurídicos**. 2021. Tese (Doutorado) — Universidade de São Paulo, 2021. Citado na página 26.

SIEKMANN, J *et al.* **Information Extraction: Towards Scalable, Adaptable Systems**. [S.l.]: Springer Science & Business Media, 1999. Citado na página 35.

SILVA, M d AO. **Pré-Processamento em Mineração de Dados como método de suporte à modelagem algorítmica**. 2014. Tese (Doutorado) — Dissertação (Mestrado)—UniversidadeFederal do Tocantins, Curso de Pós . . . , 2014. Citado na página 24.

SMART, Jignesh V. An implementation of a knowledge-based programming language with dynamic grammar. **International Journal**, v. 6, n. 5, 2016. Citado na página 24.

SOUSA, Cláudio Ribeiro de *et al.* Construção de um classificador automático de severidade de bugs para sistemas open source. Universidade Federal de Ubelândia, 2016. Citado na página 15.

SOUZA, Fábio; NOGUEIRA, Rodrigo; LOTUFO, Roberto. Portuguese named entity recognition using bert-crf. **arXiv preprint arXiv:1909.10649**, 2019. Citado na página 26.

STRAKA, Milan; HAJIC, Jan; STRAKOVÁ, Jana. Udpipe: trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing.
In: **Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)**. [S.l.: s.n.], 2016. p. 4290–4297. Citado na página 24.

TAVARES, Ricardo. Extensões da estatística scan espacial utilizando técnicas de otimização multi-objetivo. Universidade Federal de Minas Gerais, 2009. Citado na página 31.

VALENZUELA, Ricardo EG; SCHWARTZ, William R; PEDRINI, Helio. Redução de dimensionalidade aplicada à descrição de características visuais. 2011. Citado na página 53.

VANDERPLAS, Jake. **Python data science handbook: Essential tools for working with data**. [S.l.]: "O'Reilly Media, Inc.", 2016. Citado na página 33.

VASWANI, Ashish *et al.* Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017. Citado 2 vezes nas páginas 26 e 37.

VENTORIM, IGOR DE MOURA. Birchscan: Um método de aproximação do dbscan para grandes conjuntos de dados. 2021. Citado na página 41.

VIBRANS, Alexander Christian *et al.* Metodologia do inventário florístico florestal de santa catarina. **Diversidade e conservação dos remanescentes florestais: inventário florístico florestal de Santa Catarina**, v. 1, p. 31–63, 2012. Citado na página 31.

VIEIRA, Renata; LOPES, Lucelene. Processamento de linguagem natural e o tratamento computacional de linguagens científicas. **EM CORPORA**, p. 183, 2010. Citado na página 21.

WANG, Huayan; YANG, Qiang. Transfer learning by structural analogy. In: **Twenty-Fifth AAAI Conference on artificial intelligence**. [S.l.: s.n.], 2011. Citado na página 16.

WANG, Peilu *et al.* Learning distributed word representations for bidirectional lstm recurrent neural network. In: **Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**. [S.l.: s.n.], 2016. p. 527–533. Citado na página 25.

XIANG, Ruizhi *et al.* A comparison for dimensionality reduction methods of single-cell rna-seq data. **Frontiers in genetics**, Frontiers Media SA, v. 12, 2021. Citado na página 34.

YANG, Jian *et al.* Generating contextual trajectories from user profiles. In: **Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoSpatial Simulation**. [S.l.: s.n.], 2020. p. 38–47. Citado 2 vezes nas páginas 30 e 32.

ZEVAREX, Vinicius Alexandre de Oliveira; SANTOS, Carlos Henrique da Silva. Estudos das implicações do teorema de bayes na computação natural. **Revista Brasileira de Iniciação Científica**, v. 7, n. 5, p. 58–79, 2020. Citado na página 29.