UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

ABILIO CASTRO E SILVA

REALIZANDO ESTIMATIVAS DE ESFORÇO EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE UTILIZANDO MACHINE LEARNING

QUIXADÁ

2022

ABILIO CASTRO E SILVA

REALIZANDO ESTIMATIVAS DE ESFORÇO EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE UTILIZANDO MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Profª. Ma. Antonia Diana Braga Nogueira

Coorientadora: Profª. Drª. Lívia Almada Cruz

QUIXADÁ

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S578r Silva, Abilio Castro e.
Realizando estimativas de esforço em projetos de desenvolvimento de software utilizando machine learning / Abilio Castro e Silva. – 2022.
68 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2022.
Orientação: Profa. Ma. Antonia Diana Braga Nogueira.
Coorientação: Profa. Dra. Lívia Almada Cruz.

1. Estimativa de Software. 2. Software-Desenvolvimento. 3. Processamento de linguagem natural (Computação). 4. Inteligência artificial. I. Título.

CDD 005.1

ABILIO CASTRO E SILVA

REALIZANDO ESTIMATIVAS DE ESFORÇO EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE UTILIZANDO MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ____/____/_____.

BANCA EXAMINADORA

Profª. Ma. Antonia Diana Braga Nogueira (Orientadora)
Universidade Federal do Ceará (UFC)

Profª. Drª. Lívia Almada Cruz (Coorientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

José Cezar Junior de Souza Filho
Universidade Federal do Amazonas (UFAM)

Ao meu filho João Emanuel, seu nascimento significou o meu renascimento. Sua existência é a minha esperança em dias melhores.

AGRADECIMENTOS

À Deus pelo dom da existência e por ter me dado forças para concluir este curso.

À minha família, em especial a minha mãe Margarida, minha vó Zeneide e meu Tio Célio por sempre terem acreditado e me incentivado nos estudos, meu pai Sérgio que esteve presente quando precisei, a minha companheira e mãe do meu filho Joana por compartilhar a vida comigo e pelo incentivo nos últimos anos.

À Profª. Ma. Diana Braga minha orientadora neste trabalho e que por muito tempo foi nossa coordenadora de curso, considerada por muitos alunos como uma mãe, que torce pelo sucesso de seus filhos mas que também educa da maneira necessária para nos conduzir pelo melhor caminho.

À Profª. Drª. Lívia Almada minha co-orientadora neste trabalho que contribuiu de forma significativa principalmente em questões relacionadas à aprendizado de máquina.

À todos os professores do Campus de Quixadá, especialmente os que ministraram disciplinas que fiz. Todos vocês são verdadeiros mestres que possuem o poder de transformar a sociedade através da educação.

Aos amigos que fiz durante o curso e do grupo que chamamos de "Sistema de Parceiragem (SDP)": Leandro Beserra, Cícero Lima e Breno Gonzaga, os quais dividiram momentos memoráveis comigo.

Aos colegas da turma de Engenharia de Software 2017.1 que contribuíram direta e indiretamente para o meu aprendizado e em especial à: Eduardo Alves, Maryzangela Bessa, Douglas Damasceno e Jorder Paulo.

Aos colegas da bolsa PET-TI Conexões e Saberes que compartilharam comigo conhecimentos e vivências de ensino, pesquisa e extensão.

Aos servidores da UFC Campus Quixadá na pessoa de nossa querida nutricionista Ana Cláudia com seu grande coração disposto a nos ajudar em diversos momentos.

Aos servidores terceirizados na pessoa do querido Dias que nos recebia com um sorriso sincero, transmitindo boas energias pelo campus e do Renato Silva do Restaurante Universitário que também era nosso companheiro de alguns "rachas" de futebol.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“Eu sou de uma terra que o povo padece, mas não esmorece e procura vencer...”

(Patativa do Assaré)

RESUMO

A atividade de estimativa de esforço em projetos de desenvolvimento de software representa um grande desafio para os envolvidos, em relação a precisão dessas estimativas, uma vez que muitos fatores podem influenciar diretamente no tempo requerido para que as tarefas do projeto possam ser concluídas. Questões como complexidade do sistema a ser desenvolvido, experiência da equipe e tecnologias adotadas são alguns exemplos desses fatores direcionadores de custo. Dentro desse contexto, foram desenvolvidas várias técnicas de estimativa de esforço que se encaixam em abordagens como as baseadas em julgamentos de especialistas, raciocínio por analogia, modelos paramétricos e as orientadas à aprendizado. Pesquisas recentes procuram explorar a abordagem orientada à aprendizado em que técnicas de aprendizado de máquina podem ser utilizadas para treinar modelos preditivos com dados históricos de projetos anteriores. Nessa abordagem, o treinamento do modelo poderia ser realizado visando prever uma estimativa para todo o projeto ou para tarefas individuais, sendo que no segundo caso, isso pode ser realizado através do processamento de linguagem natural do texto da descrições de tarefas. Este trabalho construiu um modelo preditivo utilizando uma rede neural para estimar o esforço através da descrição textual de tarefas de projetos *Open Source*. Para adequar o texto como entrada do modelo preditivo foi necessário empregar técnicas de processamento de linguagem natural como incorporação de palavras para a transformação do texto em vetores multidimensionais. Com base na avaliação das métricas pré-definidas observou-se que o modelo foi melhor que as linhas bases estabelecidas em alguns experimentos realizados.

Palavras-chave: estimativa de esforço de software; desenvolvimento de software; processamento de linguagem natural; aprendizado de máquina.

ABSTRACT

The effort estimation activity in software development projects represents a great challenge for those involved, in relation to the accuracy of these estimates, since many factors can directly influence the time required for the project tasks to be completed. Issues such as the complexity of the system to be developed, the team's experience and the technologies adopted are some examples of these cost drivers. Within this context, several effort estimation techniques have been developed that fit approaches such as those based on expert judgments, reasoning by analogy, parametric models and learning-oriented ones. Recent research seeks to explore the learning-oriented approach in which machine learning techniques can be used to train predictive models with historical data from previous projects. In this approach, model training could be carried out in order to predict an estimate for the whole project or for individual tasks, and in the second case, this can be done through natural language processing of the text of the task descriptions. This work built a predictive model using a neural network to estimate the effort through the textual description of Open Source project tasks. In order to adapt the text as input to the predictive model, it was necessary to use natural language processing techniques, such as word embeddings to transform the text into multidimensional vectors. Based on the evaluation of the predefined metrics, it was observed that the model was better than the baselines established in some experiments carried out.

Keywords: software effort estimation; software development; natural language processing; machine learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

SDEE	*Software Development Effort Estimation*
NLP	*Natural Language Processing*
ANN	*Artificial Neural Network*
RNN	*Recurrent Neural Network*
LSTM	*Long Short-Term Memory*
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
RMSE	*Root Mean Square Error*
COCOMO	*Constructive Cost Model*
TF-IDF	*Term Frequency–Inverse Document Frequency*
SVR	*Support Vector Regression*
GNB	*Gaussian Naive Bayes*
MMRE	*Mean Magnitude of Relative Error*
BiLSTM	*Bidirectional LSTM*
API	*Application Programming Interface*
CSV	*Comma Separated Values*
JSON	*JavaScript Object Notation*
PCA	*Principal Component Analysis*

SUMÁRIO

1 INTRODUÇÃO

A estimativa de esforço é uma atividade comum dos processos aplicados em projetos de desenvolvimento de *software* e que frequentemente está relacionada com a avaliação do sucesso do trabalho realizado em um projeto. As estimativas que são definidas nos estágios iniciais do desenvolvimento de um produto podem acabar influenciando diretamente na sua qualidade final. Nesse sentido, a importância de tal atividade surge em conjunto com a dificuldade em sua condução, tendo em vista os seus muitos fatores impactantes e as diversas incertezas existentes no início de um novo projeto (SOMMERVILLE, 2011).

Dentro desse contexto, pode-se ressaltar o trabalho de Braga (2019), que, através de uma revisão sistemática da literatura, elencou 29 fatores que impactam direta ou indiretamente na estimativa de esforço em projetos de desenvolvimento de *software*. Gerenciar todos esses fatores e a complexidade inerente à estimativa de esforço em uma empresa de *software* revela-se como uma ação propensa a erros, conforme verifica Usman *et al.* (2018), em um estudo de caso da indústria, em que a subestimação de tarefas foi uma tendência nos resultados.

Algumas técnicas de estimativa de esforço apresentadas em Sommerville (2011), foram divididas em dois grandes grupos: as técnicas baseadas em experiências e a modelagem algorítmica de custos. Embora sejam observados avanços nessas técnicas, Jørgensen (2014) afirma que um tipo de técnica baseada em experiências conhecida como julgamento de especialistas ainda é a mais utilizada. Mais recentemente, o trabalho de Vera *et al.* (2019), em um recorte focado em micro e pequenas empresas de *software* do Chile, verificou que o julgamento de especialistas também era a técnica mais utilizada.

A percepção de que a adoção das estimativas feitas por especialistas continua em alta na indústria de *software* nos últimos anos e que a margem de erro dessas estimativas ainda é considerável, como pode ser observado em Faria e Miranda (2012), faz com que outras alternativas de técnicas de estimativa sejam exploradas na literatura. Com os avanços no desempenho de processadores e demais recursos computacionais a cada ano, aliado ao crescimento na produção e armazenamento de um grande volume de dados, tem-se atualmente uma forte utilização de inteligência artificial para a resolução dos mais variados tipos de problemas Lee (2019). Com isso, observa-se também o emprego das técnicas de inteligência artificial, mais frequentemente aprendizado de máquina, em várias pesquisas (Capítulo 3) com o objetivo de melhorar a assertividade das estimativas de esforço em projetos de desenvolvimento de *software*.

É comum encontrar-se pesquisas que analisam a aplicação de aprendizado de má-

quina no problema de estimativa de esforço em projetos de desenvolvimento de *software* (*Software Development Effort Estimation* (SDEE)) utilizando conjuntos de dados com características de projetos concluídos anteriormente. Conforme verificado em Dave e Dutta (2014), que apresenta os principais conjuntos de dados explorados em trabalhos que desenvolviam modelos preditivos de aprendizado de máquina com redes neurais artificiais no problema em questão. Enquanto que Phannachitta e Matsumoto (2019), trabalhando também com alguns desses mesmos conjuntos de dados, que continham características de projetos anteriores, fizeram uma comparação de 14 algoritmos amplamente usados na comunidade de ciência de dados e entre outros resultados, observaram que esses conjuntos de dados possuem pouco número de instâncias e isso influenciou diretamente na perda de generalização do modelo na previsão de dados de validação (*overfiting*).

Uma outra abordagem para tratar esse problema tem sido explorada na literatura, consistindo na análise dos títulos e descrições textuais das tarefas a serem realizadas dentro de um projeto de *software* com técnicas de processamento de linguagem natural combinadas com técnicas de aprendizado de máquina como redes neurais artificiais para a predição do esforço de uma determinada tarefa em específico ao contrário de tentar prever o esforço para todo o projeto. É o caso do trabalho desenvolvido por Ionescu *et al.* (2017), em que o modelo preditivo mostrou possuir um melhor desempenho do que métodos paramétricos clássicos de estimativa. Entretanto, a pesquisa conduzida por Tubelis (2018) trouxe grandes contribuições para esta abordagem que serão detalhadas no Capítulo 3, porém, segundo o autor os modelos desenvolvidos não se aproximaram das estimativas realizadas por julgamento de especialistas.

Contudo, percebe-se que a utilização de processamento de linguagem natural e aprendizado de máquina se apresenta como uma abordagem que ainda pode contribuir significativamente no problema de SDEE. Tendo em vista a existência de trabalhos promissores como os de Choetkiertikul *et al.* (2019) e Dan *et al.* (2020), que obtiveram resultados próximos ao das estimativas feitas por especialistas. Além disso, o volume de dados de ferramentas de gerenciamento de projetos de código aberto (*Open Source*) publicamente disponíveis representa uma vantagem dessa abordagem, favorecendo o treinamento e teste dos modelos de aprendizado de máquina. Portanto, acredita-se que faz sentido procurar novas maneiras de trabalhar o problema em questão, ainda que dentro dessa mesma abordagem. Já que isso significa não apenas reproduzir os experimentos de trabalhos anteriores para fins de validação, mas buscar por novas arquiteturas de modelos preditivos que possam servir de base para a construção de ferramentas

automatizadas de suporte a decisão na gerência de projetos de software.

Nesse sentido, o presente trabalho tem como objetivo principal a construção de um modelo de aprendizado de máquina para realizar estimativas de esforço de tarefas a partir de descrições textuais com baixa taxa de erro. E tem como objetivos específicos: construir uma base de dados com tarefas de projetos *Open Source* e disponibiliza-lá publicamente, avaliar o desempenho de diferentes arquiteturas do modelo quando aplicadas no problema em análise e verificar a viabilidade do modelo desenvolvido no suporte à decisões de estimativa de esforço.

Os capítulos seguintes estão estruturados de forma que o Capítulo 2, apresenta a fundamentação teórica, ou seja, os conceitos que são a base para a elaboração dos experimentos de aprendizado de máquina a serem conduzidos. No Capítulo 3 tem-se os trabalhos relacionados, com o intuito de evidenciar as estratégias já abordadas, as principais semelhanças e as diferenças com a abordagem realizada neste trabalho. No Capítulo 4, encontra-se a definição do método de pesquisa empregado. O Capítulo 5 descreve como o modelo preditivo foi construído. Já o Capítulo 6 contém os resultados obtidos com os experimentos realizados nesta pesquisa, e o Capítulo 7 traz as considerações finais acerca deste trabalho e também sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo pretende apresentar a base teórica utilizada na elaboração deste trabalho. Os principais conceitos a serem abordados são: estimativa de esforço de software, processamento de linguagem natural, redes neurais artificiais, regressão e métricas de avaliação.

2.1 Estimativa de Esforço de Software

A estimativa de esforço em projetos de desenvolvimento de *software* trata-se de uma análise que a partir de uma definição das funcionalidades ou outros elementos que descrevem os aspectos de um *software* a ser desenvolvido, se realize uma previsão do tempo necessário para a conclusão do *software*. Essa análise não é algo simples de ser feito corretamente, tendo em vista as mais diversas incertezas envolvidas no início de todo projeto e por essa razão, existem grandes dificuldades (Subseção 2.1.1) em se estimar esforço com alta precisão (SOMMERVILLE, 2011).

Subentende-se então que a estimativa de esforço não se trata de uma ciência exata, mas algo que pode seguir uma série de ações determinísticas responsáveis por produzirem estimativas condizentes com a realidade, assim como afirma Maxim e Pressman (2016):

> As estimativas de custo e esforço nunca serão uma ciência exata. Muitas variáveis – fatores humanos, técnicos, ambientais e políticos – podem afetar o custo final do software e o esforço necessário para desenvolvê-lo. No entanto, as estimativas de projeto de software podem ser transformadas de algo sobrenatural para uma série de etapas sistemáticas que proporcionam estimativas com um risco aceitável.

Na tentativa de diminuir essa dificuldade de realização de estimativas de esforço com uma maior precisão foram desenvolvidas algumas técnicas de estimativa ao longo dos anos. Essas técnicas segundo Sommerville (2011) podem ser classificadas em técnicas baseadas em experiências e modelagem algorítmica de custos. As técnicas baseadas em experiências realizam estimativas com base na experiência dos gerentes e desenvolvedores em projetos anteriores, uma dessas técnicas mais populares, principalmente em projetos ágeis, é o *Planning Poker*. Em relação a modelagem algorítmica de custos, a técnica mais comumente citada é o modelo COCOMO II (SOMMERVILLE, 2011). Entretanto, em Heemstra (1992) são mencionadas mais algumas técnicas, como estimativas baseadas em: raciocínio por analogia, preço a ganhar e capacidade disponível. A categoria da técnica de estimativa utilizada neste trabalho é a orientada à aprendizado que embora ainda esteja se estabelecendo frequentemente vêm sendo mencionada na literatura como se observa na Subseção 2.1.3.

2.1.1 Dificuldades durante a atividade de estimativa de esforço de software

As dificuldades que a maioria dos projetos de desenvolvimento de *software* enfrentam quando perpassam pela atividade de estimativa de esforço podem ser listadas, assim como em Heemstra (1992). A seguir, são apresentadas 11 dificuldades elencadas por Heemstra (1992) realizando adaptações em algumas delas para se encaixarem melhor em relação aos processos e a realidade do desenvolvimento de *software* nos dias atuais.

1. Existe uma falta de dados para projetos de softwares concluídos. Esses dados poderiam apoiar as estimativas no gerenciamento de projetos. Entretanto, muito dessa falta de dados é decorrente de questões de privacidade de empresas, pois uma quantidade significativa de dados de projetos *Open Source* estão publicamente disponíveis;
2. As estimativas são realizadas apressadamente e em muitos casos mesmo sem o conhecimento completo dos requisitos dos sistema a ser desenvolvido;
3. Especificações claras, completas e confiáveis são difíceis de serem formuladas principalmente no começo do projeto. A mudança nas especificações é algo contemplado nas metodologias de desenvolvimento ágeis amplamente utilizadas atualmente;
4. Características do *software* e do desenvolvimento de *software*, tais como nível de abstração, complexidade, mensurabilidade de produto e processo, aspectos de inovação e etc. dificultam as estimativas;
5. Os fatores de custo como complexidade do *software*, compromisso e participação do cliente ou usuário e a experiência do time de desenvolvimento são difíceis de se determinar na prática;
6. A natureza de mudança constante em tecnologia da informação e nos métodos de desenvolvimento de *software* são um problema para a estabilização do processo de estimativa;
7. O estimador pode não ter muita experiência na realização de estimativas principalmente em projetos grandes, pois afinal é inviável participar de muitos projetos grandes, por exemplo, em um curto período de tempo;
8. Os desenvolvedores de *software* possuem certo viés de subestimação e em muitos casos acabam espalhando a estimativa de uma parte do sistema para todo o restante, desconsiderando questões do processo de desenvolvimento como

coordenação e gerenciamento;

9. Em alguns projetos pode ocorrer dos estimadores definirem o tempo para o desenvolvimento de uma tarefa como se eles fossem realizar tal tarefa, ignorando o fato de que alguém com menos experiência possa ser o responsável por realizar o trabalho;
10. Existe uma suposição errada de uma relação linear entre o número de recursos necessários e o tempo para realização do trabalho. Por exemplo, o *software* desenvolvido por 25 pessoas em dois anos seria desenvolvido em apenas um se a equipe fosse de 50 pessoas. Na realidade, verifica-se que adicionar mais pessoas a um projeto já atrasado só irá atrasá-lo mais ainda;
11. O estimador tende a reduzir a estimativa de certo modo para fazer com que a aposta seja mais aceitável.

2.1.2 Pré-requisitos para a atividade de estimativa de esforço de software

Antes da realização dos processos de estimativa de esforço de *software* pode-se levar em consideração diversos fatores que irão influenciar nas decisões relacionadas com essa atividade. No que se refere ao ponto de vista organizacional, observam-se questões de gerenciamento de projeto como: alocação de responsabilidades, tomada de decisão, organização do trabalho do projeto, monitoramento e auditoria do desenvolvimento de tarefas. Partindo dos aspectos sociológicos e psicológicos da estimativa de esforço de *software* encontra-se questões voltadas para o compromisso, a organização da coesão do grupo, o estilo da liderança etc. O ponto de vista técnico do trabalho também tem sua importância ao lidar com temas como: a disponibilidade de bons recursos como ferramentas de *design*, programação, teste e documentação, facilidades de *hardware* etc (HEEMSTRA, 1992).

Visando tratar esses fatores mencionados acima Heemstra (1992) afirma que devem ser preenchidos pré-requisitos para garantir uma base sólida para prever esforço, duração e a capacidade para desenvolver o *software*. Esses pré-requisitos são:

Percepções nas características de:

- Produto (*software*) a ser desenvolvido (O QUÊ);
- Meios de produção (COM O QUÊ);
- Pessoal de produto (QUEM);
- Organização do trabalho (COMO);

- Usuário/Organização do usuário (PARA QUEM).

Disponibilidade de:

- Técnicas e ferramentas para estimativa de custo de *software* (Subseção 2.1.3).

Ainda segundo Heemstra (1992) existem muitos fatores direcionadores de custos citados pela literatura. Portanto, cabe a organização desenvolvedora de *software* observar quais são esses fatores mais dominantes. Dessa forma, o Quadro 1 apresenta os pré-requisitos de percepções de características (O QUÊ, COM O QUÊ, QUEM, COMO, PARA QUEM) relacionados com os fatores direcionadores de custos que segundo Heemstra (1992) são os mais importantes.

Quadro 1 – Pré-requisitos e fatores direcionadores de custo

O QUÊ (produto)	**COM O QUÊ (meios)**	**QUEM (pessoal)**	**COMO (projeto)**	**PARA QUEM (usuário)**
Tamanho do *software* Qualidade requerida Volatilidade de requisitos Complexidade do *software* Nível de reúso Quantidade de documentação Tipo de aplicação	Restrições de computação: - tempo de execução; - tempo de resposta; - capacidade de memória. Uso de ferramentas Uso de técnicas avançadas de programação	Qualidade do pessoal Experiência do pessoal Gerenciamento de qualidade Disponibilidade para o projeto	Duração dos requisitos de projeto: - alongamento; - compressão. Bases para controle de projeto: - matriz organizacional; - organização do projeto - prototipação - incremental - desenvolvimento linear - desenvolvimento de *software*	Participação Número de usuários Estabilidade da organização do usuário, procedimentos, maneiras de trabalhar Experiência do usuário com automação, nível de educação em automação

Fonte: adaptado de (HEEMSTRA, 1992).

Conforme observa-se na literatura e na prática, não existe facilidade no tratamento dos fatores direcionadores de custo. Além do difícil trabalho de escolher os mais importantes para determinada situação, é comum encontrar-se problemas ao mensurar tais fatores e sua real influência para a estimativa de esforço.

2.1.3 Técnicas e ferramentas para a atividade de estimativa de esforço de software

Esta subseção pretende apresentar cada categoria de técnica de estimativa de esforço citada anteriormente. Porém, é importante mencionar antes que a estimativa de esforço pode ser observada na perspectiva de duas abordagens diferentes como afirma Heemstra (1992):

> 1. *Top-Down*: A estimativa do projeto no geral é derivada das características globais do produto. O custo total estimado é então dividido entre os vários componentes.
>
> 2. *Bottom-Up*: O custo de cada componente individual é estimado pela pessoa que será responsável pelo desenvolvimento do componente. Os custos individuais são somados para obter o custo geral estimado do projeto.

Embora não exista um consenso ou padronização acerca das categorias de estimativa de esforço é comum encontrar-se na literatura o agrupamento de técnicas de estimativa de esforço da seguinte forma:

1. Estimativas baseadas em julgamento de especialistas: um novo projeto que utiliza essa categoria de técnica confia no nível de habilidade e experiência que um especialista tem em relembrar fatos de projetos anteriores. A maiorias dessas estimativas são qualitativas e não objetivas. Um problema comum desse método é a dificuldade que alguém pode ter em realizar o trabalho no tempo definido pelo especialista caso este não leve em consideração o nível de experiência da pessoa que desenvolverá a atividade de trabalho (HEEMSTRA, 1992).
2. Estimativas baseadas em raciocínio por analogia: o princípio dessa técnica está centrada na análise de dados históricos de projetos anteriores ou partes (módulos) de projetos anteriores que possuem similaridades com o novo projeto a ser desenvolvido. Coletar os dados referentes às características de projetos anteriores pode dificultar a adoção dessa técnica (HEEMSTRA, 1992).
3. Estimativas baseadas em preço a ganhar: razões comerciais fazem com que a estimativa do *software* seja baseada no preço para ganhar o projeto. Ou seja, a estimativa é baseada no orçamento do cliente. Esse tipo de estimativa é colocado por Heemstra (1992) não como uma técnica de estimativa em si, entretanto, este mesmo autor afirma que organizações que usam essa estimativa não possuem menos acurácia na estimativa do que organizações que utilizam outros métodos. Enquanto que Borade e Khalkar (2013) ressaltam que este método não é uma boa prática uma vez que pode levar à atrasos na entrega e forçar situações em

que o time de trabalho tenha que realizar horas extras. Por exemplo, um *software* que deveria ser desenvolvido por 100 pessoas por mês, mas que o cliente só possui orçamento para 60 pessoas por mês faz com que as estimativas sejam ajustadas a esse número de 60 pessoas por mês para ganhar o projeto (BORADE; KHALKAR, 2013).

4. Estimativas baseadas em capacidade disponível: a base desse método leva em consideração os recursos disponíveis na organização, principalmente pessoal. Por exemplo, se existem 3 pessoas disponíveis para um novo projeto nos próximos quatro meses então a estimativa será de 12 pessoas por mês. O problema desse método acontece em situações de superestimação em que o esforço planejado será usado completamente, seguindo assim à lei de Parkinson que estabelece o seguinte: *"o trabalho se expande de modo a preencher o tempo disponível para a sua realização."* (HEEMSTRA, 1992).
5. Estimativas baseadas no uso de modelos paramétricos: o tempo de desenvolvimento é estimado em função de um número de variáveis. Sendo que estas variáveis representam os mais importantes fatores direcionadores de custo. Os modelos de estimativa são formados principalmente de um número de algoritmos e parâmetros. Os valores dos parâmetros e os tipos de algoritmos a serem utilizados nos modelos são baseados no conteúdo de um conjunto de dados de projetos completados. Alguns exemplos desses modelos são: COCOMO, Putnam e Price-S (HEEMSTRA, 1992). Para mais detalhes sobre esses e outros modelos clássicos o trabalho de Abbas *et al.* (2012) pode ser consultado.
6. Estimativas orientadas à aprendizado: apesar da categoria de estimativa baseadas em raciocínio por analogia ser colocada por Borade e Khalkar (2013) como um método que faz parte da estimativa orientada à aprendizado, faz mais sentido englobar nesta categoria as técnicas de estimativa que usam modelos e técnicas oriundas da inteligência artificial como redes neurais e lógica difusa. Nesse tipo de técnica modelos preditivos são construídos e treinados com dados históricos de projetos anteriores, essas informações podem ser sobre os projetos concluídos como um todo ou relativas a tarefas específicas de cada projeto onde pode-se aliar a utilização de processamento de linguagem natural para a análise de descrições textuais de tarefas (BARDSIRI; HASHEMI, 2014).

2.2 Processamento de Linguagem Natural

Uma definição frequentemente utilizada sobre processamento de linguagem natural (*Natural Language Processing* (NLP)) é a de Liddy (2001):

> Processamento de Linguagem Natural é um conjunto de técnicas computacionais teoricamente motivadas para analisar e representar ocorrências naturais de texto em um ou mais níveis de análise linguística para o propósito de se obter um processamento de linguagem semelhante ao humano para um um grupo de tarefas ou aplicações. (Tradução livre)

Esta definição implica que as técnicas de NLP podem tomar proveito do poder de processamento computacional atualmente disponível para analisar a linguagem natural que as pessoas usam para se comunicar diariamente, seja por fala ou através de texto. Sendo que esta análise poderá envolver elementos da análise linguística como: fonética, morfologia, sintaxe, semântica, dentre outros. Esses elementos são referidos na definição de Liddy (2001) como níveis de análise linguística.

Atualmente, as teorias e técnicas de NLP possuem muitas implementações que fazem parte da construção de diversos tipos de aplicações em vários sistemas operacionais e dispositivos eletrônicos. Muitos desses avanços são resultados da combinação de NLP com outras áreas da Inteligência Artificial, como Aprendizado de Máquina e sua sub-área de aprendizado profundo (*Deep Learning*), como mostra o trabalho de Torfi *et al.* (2020). Algumas das aplicações das técnicas de NLP comumente reconhecidas são: autocorreção de texto, autocomplemento de frases, tradução de línguas, *chatbots*, reconhecimento de fala, assistentes de voz, corretores gramaticais e classificadores de texto.

O presente trabalho utilizará a técnica de NLP denominada incorporação de palavras (*Word Embeddings*) que também se insere nas técnicas de Aprendizado de Máquina e por esse motivo será detalhada na Subseção 2.3.1. Além disso, a limpeza dos dados textuais do conjunto de dados utilizado nos experimentos é considerada uma técnica de NLP e o modelo preditivo que realizará estimativas de esforço consiste em um modelo de regressão.

2.3 Aprendizado de Máquina

A definição inicial do termo Aprendizado de Máquina geralmente é creditada à Samuel (1959) que realizou esforços em uma abordagem utilizando jogos para simular em computador um comportamento que se realizado por um humano ou outro animal envolveria

de certa forma algum processo de aprendizagem. Segundo o trabalho de Samuel (1959) seria possível que um computador fosse programado para aprender a jogar damas e se saísse melhor no jogo do que as pessoas que escreveram o programa (SAMUEL, 1959).

Uma outra definição para o termo seria o fato de se considerar que um determinado agente estaria aprendendo se a avaliação da sua performance em uma tarefa melhorasse com o tempo através da própria observação das informações disponíveis. Uma forma de se exemplificar essa definição poderia ser observando pares de valores `(x, y)` onde `x` é a entrada para uma função `f` desconhecida em que `y = f(x)` e o algoritmo que, para novos valores de `x` conseguisse realizar uma previsão de `y` com o valor aproximado ao da função `f` estaria então aprendendo a se comportar como a própria função `f` (RUSSELL; NORVIG, 2013).

As aplicações de Machine Learning nos dias atuais cresceram significativamente e começam a alcançar áreas fora das ciências da computação como medicina e biologia (CHING *et al.*, 2018). E o presente trabalho pretende aplicar algumas técnicas de Machine Learning, que serão detalhadas nas subseções seguintes, visando a melhoria de parte de um processo de engenharia de software que nesse caso é a estimativa de esforço.

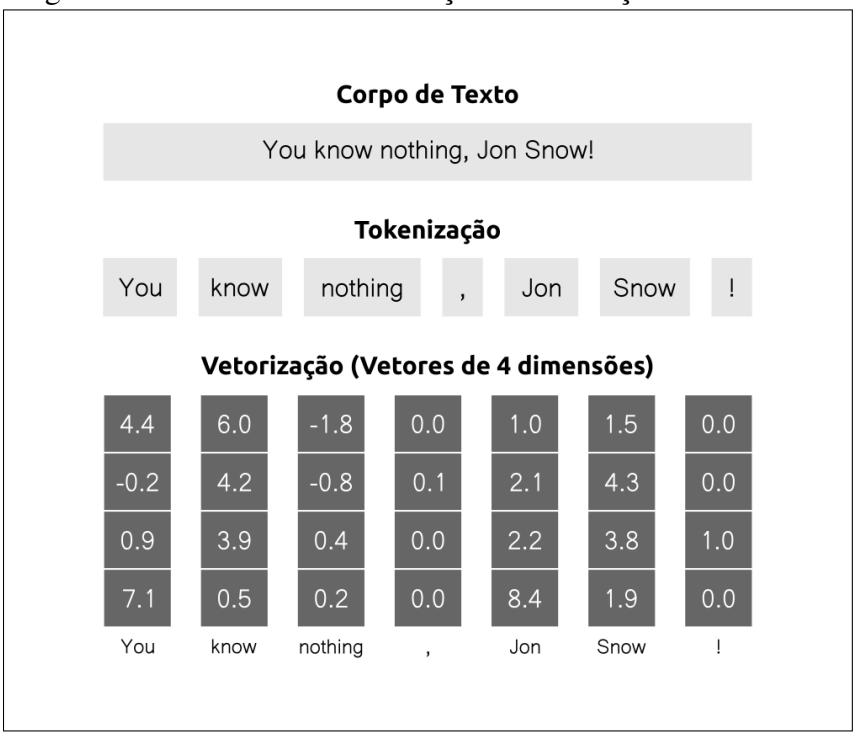
2.3.1 Incorporação de Palavras

A técnica de *Word Embeddings* permite que se faça uma representação de unidades de um corpo de texto como vetores de múltiplas dimensões contendo valores numéricos de ponto flutuante. Essas unidades derivadas do corpo de texto podem ser palavras, sequências de palavras, ou caracteres e são conhecidas como *tokens*, enquanto que o processo de separar o texto em *tokens* é chamado de *tokenização*. Após a *tokenização* pode-se então realizar a vetorização desses *tokens* que é a tranformação de um ou mais caracteres que formam um *token* em um vetor numérico. Os passos citados para a representação do corpo de texto em vetores são exemplificados na Figura 1 (CHOLLET, 2018).

Um exemplo clássico de operações utilizando vetores de *Word Embeddings* é a operação de subtrair do vetor da palavra rei o vetor da palavra homem e depois adicionar o vetor da palavra mulher, o resultado deve ser um vetor muito próximo do vetor da palavra rainha. Essa relação mostrada na Figura 2 pode ser interpretada como: a palavra rei está para a palavra rainha assim como a palavra homem está para a palavra mulher (CHOLLET, 2018).

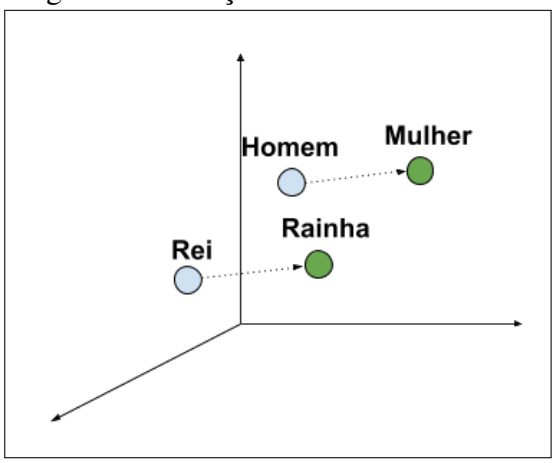
A conversão de texto para vetor permite que sejam desenvolvidos modelos de *Word Embeddings* que, dependendo da sua arquitetura, podem aprender sobre a semântica do corpo

Figura 1 – Processos de tokenização e vetorização de texto



Fonte: elaborado pelo autor.

Figura 2 – Relações entre vetores



Fonte: (CONTRATRES, 2018)

de texto fornecido. Uma vez treinados esses modelos possibilitam a execução de operações envolvendo palavras como obter o percentual de similaridade entre duas palavras ou dado uma palavra quais são as suas `n` palavras mais similares. A utilização de *Word Embeddings* em problemas de classificação de texto também é importante pois os modelos de aprendizado de máquina não recebem texto como entrada mas sim a sua representação vetorial (CHOLLET, 2018).

Existem duas maneiras de se obter um modelo de *Word Embeddings*: aprender um novo modelo com o corpo de texto fornecido ou utilizar *Word Embeddings* pré-treinadas. A

primeira opção parte do princípio de se inicializar um vetor que representa uma palavra, por exemplo, com números aleatórios e conforme o modelo é atualizado os valores desses números no vetor serão modificados para refletir a semântica aproximada da palavra em relação ao contexto que ela está inserida dentro do domínio do corpo de texto. A segunda maneira consiste em carregar *Word Embeddings* pré-treinadas para outras tarefas sendo que em muitos casos, essas tarefas ou o domínio do texto delas compreendem questões mais gerais como o vocabulário da Wikipédia ou das notícias do Google News (CHOLLET, 2018).

A especificidade do domínio do problema abordado neste trabalho que está inserida no contexto do desenvolvimento profissional de *software* contribui para que a criação de um novo modelo de *Word Embeddings* que aprende a partir do corpo de texto passe a ser a opção escolhida para a implementação dos experimentos que serão realizados. Entretanto, ainda se faz necessário avaliar o desempenho de *Word Embeddings* pré-treinadas no problema para validar ou não o uso da outra abordagem citada.

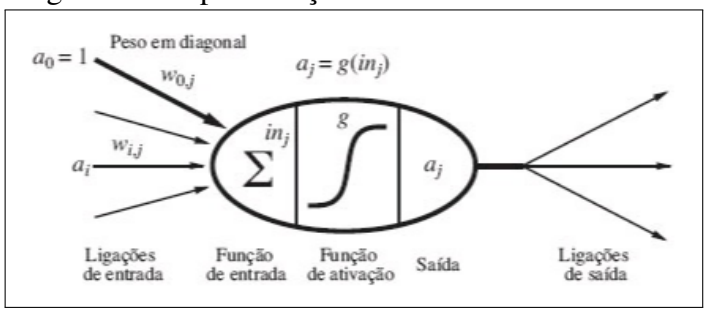
2.3.2 Redes Neurais Artificiais

As redes neurais artificiais (*Artificial Neural Network* (ANN)) são modelos computacionais que, inspirados por hipóteses de pesquisas da neurociência acerca do funcionamento do cérebro animal, realizam processamentos nos dados de entrada fornecidos para a rede e geram dados de saídas que são utilizados para se inferir respostas sobre problemas específicos para os quais o modelo foi treinado (RUSSELL; NORVIG, 2013).

O bloco básico que compõe uma ANN é chamado de neurônio em razão de uma analogia às células homônimas que fazem parte do sistema nervoso animal. Um neurônio basicamente é composto por ligações de entrada, função de entrada, função de ativação, saída e ligações de saída conforme a representação da Figura 3. As ligações de entrada recebem valores que são agregados pela função de entrada em uma soma ponderada e o resultado dessa soma é aplicado à função de ativação para obter a saída do neurônio. As ligações possuem pesos usados para definir o valor da importância de uma conexão específica entre dois neurônios e também faz parte do cálculo da soma ponderada da função de entrada (RUSSELL; NORVIG, 2013).

As conexões entre os neurônios definidos anteriormente irão formar a rede neural propriamente dita. O arranjo dessas conexões geralmente é feito em camadas onde as entradas de um neurônio de uma camada se originam em neurônios da camada anterior. Podem existir redes com apenas uma camada que são conhecidas como *perceptron* ou redes com múltiplas

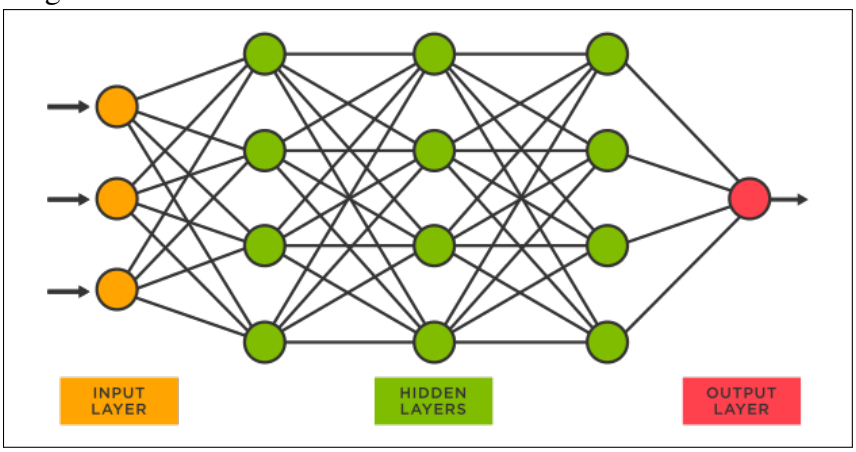
Figura 3 – Representação de um neurônio artificial



Fonte: (RUSSELL; NORVIG, 2013)

camadas contendo camadas ocultas que não estão diretamente ligadas a saída da rede. A Figura 4 apresenta como seria o arranjo de uma rede neural multicamadas contendo uma camada de entrada em laranja com 3 neurônios, 3 camadas ocultas cada uma com 4 neurônios em verde e uma camada de saída com um neurônio em vermelho (RUSSELL; NORVIG, 2013).

Figura 4 – Rede neural artificial multicamadas



Fonte: (TIBCO, 2021)

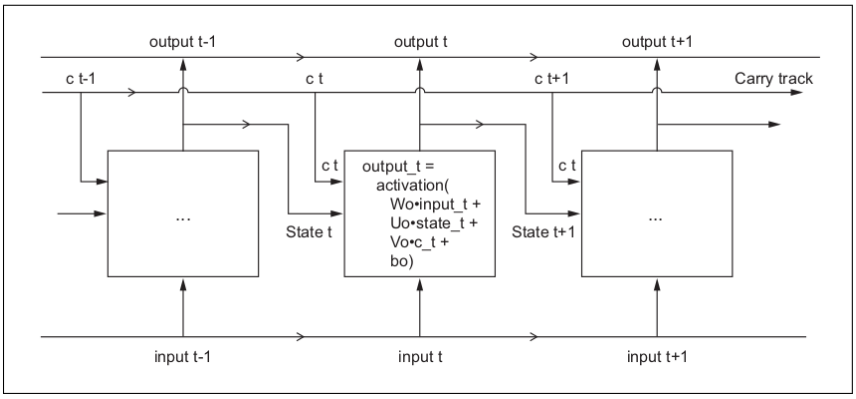
Outro aspecto importante das redes neurais é a direção das conexões entre os seus neurônios. Através dessa propriedade pode-se definir dois tipos de redes neurais: com alimentação para frente e a recorrente. Na rede neural com alimentação para frente os neurônios de uma camada recebem a entrada de uma camada anterior e enviam sua saída para neurônios da camada posterior formando um grafo acíclico dirigido. A rede neural recorrente contém neurônios que podem receber como entrada a sua própria saída e essa característica fornece a esse tipo de rede uma memória de curto prazo permitindo que a função de ativação também considere informações anteriores para calcular a saída (RUSSELL; NORVIG, 2013).

Acredita-se que para um melhor entendimento do contexto das palavras contidas em uma tarefa que se precisa estimar o tempo de desenvolvimento, as redes neurais recorrentes surgem como a opção mais viável. A subseção seguinte pretende apresentar mais detalhes sobre

essa arquitetura de rede neural e qual variação dela será utilizada neste trabalho.

As redes neurais artificiais recorrentes (*Recurrent Neural Network* (RNN)) são consideradas por autores como Russell e Norvig (2013) como uma representação mais próxima do modelo cerebral humano, porém, mais difíceis de compreender. O tipo mais básico de RNN apenas recebe o estado do neurônio anterior, a entrada atual e aplica uma função de ativação nesses valores para produzir a saída. Enquanto que outro tipo de RNN conhecida como Memória de Longo-Curto Prazo (*Long Short-Term Memory* (LSTM)) é estruturada de forma que seus neurônios possuem portas internas que permitem um maior gerenciamento do armazenamento das informações que chegam através da entrada atual, da saída do neurônio anterior e também de uma entrada que mantém um estado anterior de longo prazo. A Figura 5 mostra a estrutura de uma rede neural LSTM em que `input_t` é a entrada atual, `state_t` é o valor da saída do neurônio anterior e `c_t` é o estado anterior de longo prazo (*Carry Track*) (CHOLLET, 2018).

Figura 5 – Estrutura de uma rede neural LSTM



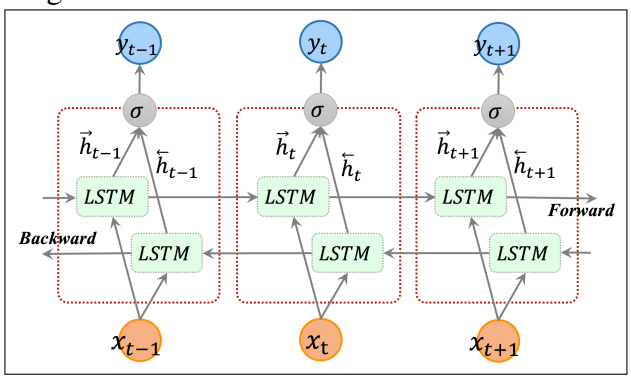
Fonte: (CHOLLET, 2018)

Uma melhoria da LSTM permite que se leve em conta na análise não apenas memórias anteriores da sequencia mas também as próximas memórias. Essa melhoria é chamada de BiLSTM e pode ser obtida com a utilização de uma segunda cadeia de LSTM no sentido contrário ao da primeira cadeia. A Figura 6 apresenta a disposição de uma rede neural LSTM Bidirecional onde se pode observar uma cadeia de LSTM para frente (*Forward*) e outra para trás (*Backward*) (CHOLLET, 2018).

2.3.3 Regressão e Métricas de Avaliação

Em aprendizado de máquina a técnica de regressão consiste na previsão de valores numéricos contínuos, diferentemente da classificação que procura prever rótulos ou categorias como saída do modelo. Uma vez que o objetivo do modelo a ser desenvolvido é predizer um valor

Figura 6 – Rede neural bidirecional LSTM



Fonte: (CUI *et al.*, 2018)

contínuo de tempo em minutos, faz sentido utilizar-se da regressão para este fim (HARRISON, 2019).

Em relação à avaliação dos resultados dos modelos de regressão, existem várias métricas oriundas da estatística, com suas especificidades distintas, que podem ser utilizadas para tanto. Nesse sentido, duas dessas métricas foram escolhidas para a avaliação do modelo durante o seu treinamento e também no resultado dos testes.

Uma das métricas utilizadas será a Média do Erro Absoluto (*Mean Absolute Error* (MAE)) definida na Equação 2.1. Em que y_{real} é o valor de saída real para uma entrada do dataset e y_{pred} é o valor previsto pelo modelo.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_{real} - y_{pred}| \tag{2.1}$$

Outra métrica utilizada para avaliação será a Média do Erro Quadrático (*Mean Squared Error* (MSE)) que é definida pela Equação 2.2.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_{real} - y_{pred})^2 \tag{2.2}$$

A diferença entre as métricas MAE e MSE consiste na penalização maior do erro obtida na MSE, pois esta eleva o erro absoluto ao quadrado e por essa sensibilidade maior ao erro contribui de forma efetiva no treinamento e consequentemente melhoria do modelo. Além dessas duas métricas, também será utilizado a métrica Raiz Média do Erro Quadrático (*Root Mean Square Error* (RMSE)) que nada mais é do que a raiz quadrada da métrica MSE.

3 TRABALHOS RELACIONADOS

Nas subseções a seguir, serão apresentados quatro trabalhos relacionados com este trabalho e na última subseção será apresentado um quadro com as principais características desses trabalhos de forma que será possível visualizar melhor quais são as diferenças e os pontos em comum com este trabalho. A principal característica em comum entre os trabalhos relacionados com esta pesquisa é a utilização de descrições textuais de tarefas de projetos de desenvolvimento de *software* como entrada para treinamento dos modelos de aprendizado de máquina. Uma vez treinados, esses modelos podem fazer previsões de estimativas de esforço para uma determinada nova tarefa não utilizada no treinamento.

3.1 Natural Language Processing and Machine Learning Methods for Software Development Effort Estimation

O trabalho desenvolvido em Ionescu *et al.* (2017) se apresenta como um dos pioneiros na construção de modelos que utilizam processamento de linguagem natural e aprendizado de máquina para conduzir estimativas de tarefas de desenvolvimento de software a partir de suas descrições textuais.

Para tanto, após a introdução do contexto do problema abordado e da motivação do desenvolvimento do trabalho Ionescu *et al.* (2017) fazem uma pequena revisão de algumas técnicas de estimativa de esforço existentes, apresentando técnicas baseadas em experiências mais atuais como *Planning Poker* e técnicas baseadas em algoritmos mais clássicas como o modelo construtivo de custo (*Constructive Cost Model* (COCOMO)).

As técnicas elencadas por Ionescu *et al.* (2017) e que foram utilizados nos experimentos do trabalho englobam temas como: Frequência do Termo - Inverso da Frequência no Documento (*Term Frequency–Inverse Document Frequency* (TF-IDF)), representação distribuída de documentos (usando doc2vec), regressão de vetores de suporte (*Support Vector Regression* (SVR)) e Naive Bayes Gaussiana (*Gaussian Naive Bayes* (GNB)).

Os dados utilizados na pesquisa de Ionescu *et al.* (2017) foram fornecidos por um empresa que desenvolvia *software* e também tarefas de manutenção geral em tecnologia da informação. Esses dados foram divididos em um conjunto de dados 'T' que continha tarefas representadas seguindo um modelo (*template*) e outros oito conjuntos de dados 'd_i' que continham tarefas não relacionadas a programação e que eram representadas de forma mais simples, ou seja,

não seguiam um *template*.

O método de pesquisa em Ionescu *et al.* (2017) consistiu em tratar o conjunto de dados 'T' de forma diferente dos conjuntos de dados 'd_i'. O fluxo de treinamento dos modelos com o conjunto de dados 'T' possuía um pré-processamento que concatenava a descrição textual com outros atributos antes de seguir para o próximo estágio que era a representação do texto em vetores onde se utilizou para isso Frequência do Termo - Inverso da Frequência no Documento e representação distribuída de documentos. O passo final tratava de fornecer os dados para modelos que poderiam aprender com as relações do texto e o tempo real para conclusão da tarefa. O método proposto utilizou um modelo SVR, mas também realizou testes com um modelo GNB. Além disso, um estágio de busca de hiperparametrização foi feito para encontrar as melhores configurações de parâmetros para os modelos que seriam avaliados com a métrica Média da Magnitude do Erro Relativo (*Mean Magnitude of Relative Error* (MMRE)).

Os resultados dos experimentos executados em Ionescu *et al.* (2017) mostraram que as melhores médias de MMRE foram obtidas usando TF-IDF como representação de texto em vetor e SVR como algoritmo. Segundo os autores, o treino do modelo também acontecia mais rápido utilizando TF-IDF e, além disso, afirmaram que doc2vec se saia melhor em problemas de classificação. Outro aspecto interessante dos resultados, foram as melhores médias dos experimentos serem encontradas no conjunto de dados 'T' confirmando que uma melhor estruturação das descrições das tarefas podem levar à estimativas melhores.

3.2 Estimating Software Development Task Effort Using Word Embeddings and Recurrent Neural Networks

O trabalho desenvolvido por Tubelis (2018) consistia em construir um modelo de rede neural que estimava tarefas de desenvolvimento de software, a partir de descrições textuais em inglês, com um desempenho significativamente melhor do que a média e a mediana do esforço real das tarefas.

Segundo Tubelis (2018), as estimativas de esforço são realizadas na maioria das vezes pelos gerentes de projeto e desenvolvedores apenas com base em experiências, enquanto que métodos paramétricos e com uso intensivo de dados são menos populares. Como trabalhos relacionados, Tubelis (2018) cita seus próprios trabalhos desenvolvidos anteriormente e um trabalho anterior de Choetkiertikul *et al.* (2019), que junto com o código-fonte disponibilizado no *GitHub* serviu como inspiração para a pesquisa desenvolvida.

A composição dos conjuntos de dados em Tubelis (2018) foi feita com a busca por repositórios de projetos *Open Source* que usavam a ferramenta de rastreio de tarefas Jira. Uma vez encontrados esses repositórios, poderia-se então buscar e armazenar as tarefas contidas nesses repositórios. Com esse método, foi possível montar um conjunto de dados para treinamento com 32 repositórios e mais de 63000 tarefas com tempo de desenvolvimento reportado. Para validação, através de um acordo de confidencialidade feito com duas empresas de software, foi possível montar um conjunto de dados com mais de 35000 tarefas.

A pesquisa de Tubelis (2018) foi conduzida por um método de 3 fases. Na primeira fase, os dados eram obtidos; a segunda fase consistia de vários ciclos de projeto e experimentação, sendo que, em cada ciclo, um novo modelo era construído com melhorias do modelo anterior e, finalmente, na última fase, o modelo era avaliado com o conjunto de dados de validação obtido com as empresas de software.

Embora Tubelis (2018) tenha alcançado o objetivo principal com os resultados dos experimentos, apresentando Média do Erro Absoluto (MAE) melhores do que as estimativas usando a média e a mediana, a resposta a pergunta de pesquisa sobre a possibilidade do modelo ser usado em contextos comerciais mostrou que o modelo desenvolvido possui uma precisão inferior a estimativa feita por humanos segundo o autor. Entretanto, essa pesquisa foi uma grande contribuição para as pesquisas futuras no problema de SDEE, principalmente, na obtenção de dados, tendo em vista, os algoritmos publicamente disponíveis no *GitHub* que são boas ferramentas para a busca e estruturação dos dados de repositórios Jira públicos.

3.3 A Deep Learning Model for Estimating Story Points

A pesquisa realizada por Choetkiertikul *et al.* (2019) tem por objetivo propor um novo modelo para estimar pontos de estória (uma unidade de medida de esforço) a partir das descrições textuais das tarefas. O novo modelo seria então utilizado como ferramenta de suporte para a estimativa humana.

A revisão da literatura em Choetkiertikul *et al.* (2019) é bem estruturada. Apresenta a classificação dos métodos de estimativa de esforço comumente utilizados dividindo as técnicas em baseadas em experiências, baseadas em modelo e também vai além ao citar uma abordagem híbrida que seria uma combinação de julgamento de especialistas em conjunto com os dados disponíveis. Essa abordagem híbrida segundo os autores se assemelha como a forma de utilização sugerida para o novo modelo proposto que é desenvolvido no trabalho. Vários trabalhos

relacionados e suas técnicas utilizadas para resolver o problema de SDEE são citados, assim como, as diferentes aplicações para o tipo de rede neural artificial (ANN) usada pelo os autores: Memória de Curto e Longo Prazo ou LSTM.

A base teórica da pesquisa de Choetkiertikul *et al.* (2019) concentra-se nos temas: estimativa de pontos de estória, redes neurais artificiais como a memória de curto-longo prazo (LSTM) que é um tipo de rede neural recorrente e também incorporação de palavras como a técnica para representação de palavras como vetor.

Diferentemente de Tubelis (2018), onde se faz uma busca automatizada para encontrar os repositórios Jira públicos, em Choetkiertikul *et al.* (2019), são escolhidos 9 repositórios que possuem esforço reportado em pontos de estória e que os autores consideram como os maiores repositórios de projetos *Open Source*. Os repositórios coletados possuem vários projetos mas para a construção do conjunto de dados só foram considerados projetos com mais de 300 tarefas. Além disso, também foi aplicado um filtro para que tarefas com 0 pontos de estória (erros de software não-reproduzíveis), ou aquelas que possuíam pontos de estórias fora da realidade (valores negativos ou maiores que 100). Com isso, no total foi possível criar um conjunto de dados com 23313 tarefas de 16 diferentes projetos, correspondendo a apenas 2,66% do total das tarefas coletadas.

Embora não tenham explicitamente definido um método para o desenvolvimento do modelo apresentado percebe-se que Choetkiertikul *et al.* (2019) construíram uma arquitetura inicial e foram experimentando novas configurações para os parâmetros. Dessa maneira, foi possível realizar uma sintonia dos diversos paramêtros do modelo.

Da mesma forma que em Tubelis (2018), a média do erro absoluto do modelo em Choetkiertikul *et al.* (2019) foi superior ao resultado da média e mediana das estimativas. Outro resultado interessante observado na pesquisa foi que as estimativas realizadas com projetos de diferentes repositórios (como por exemplo, treinamento do modelo com um projeto de um repositório e estimativas realizadas com um projeto de um outro repositório) fazem com que o erro do modelo cresça. Os autores também apresentam um melhor desempenho do modelo em relação ao trabalho desenvolvido por Porru *et al.* (2016) que também faz estimativas de tarefas utilizando a unidade de medida de esforço conhecida como pontos de estória. Contudo, os autores consideram os resultados obtidos como significativos e que o uso das técnicas estudadas no trabalho podem ser promissores para a acadêmia e o mercado de software.

3.4 An NLP Approach to Estimating Effort in a Work Environment

O trabalho de Dan *et al.* (2020) tem como principal objetivo construir um sistema capaz de estimar tarefas e comparar os resultados da estimativa desse sistema com os resultados da estimativa humana.

Na fundamentação teórica de Dan *et al.* (2020) alguns métodos clássicos de estimativa de esforço de software, tais como, análise de pontos de função e COCOMO são citados, assim como, uma menção para o trabalho de Matson *et al.* (1994) que aborda os problemas existentes nesses métodos. Além disso, os autores afirmam que tais métodos se baseiam principalmente na estimativa do tamanho do software, que deve ser feita antes de se utilizar esses métodos, entretanto, estimar o tamanho de um software também não é uma tarefa fácil, tendo em vista que, segundo os autores, a real viabilidade do uso do tamanho de um software (em linhas de código) para a estimativa final pode ser debatida.

As técnicas utilizadas na pesquisa de Dan *et al.* (2020) foram: incorporação de palavras (*word embeddings*) para representação do texto em vetor e memória de curto-longo prazo bidirecional, *Bidirectional LSTM* (BiLSTM), como a rede neural do modelo. Além disso, foi utilizado uma técnica de processamento de linguagem natural para reduzir as palavras aos seus radicais conhecida como *stemming*. O BiLSTM, conforme os autores, seria melhor em entender o contexto pois conseguiria guardar informações do passado, como acontece no LSTM e também informações do futuro através da camada adicional.

Os dados para treinamento e validação do modelo desenvolvido por Dan *et al.* (2020) foram fornecidos por uma empresa. Além da descrição textual da tarefa, cada entrada do conjunto de dados utilizado possuía metadados como: cliente, projeto, nível de experiência do desenvolvedor (júnior, pleno, sênior), tipo de desenvolvedor (interno ou externo) e número de horas trabalhadas na tarefa. Os dados passaram por processos de limpeza e a coluna da descrição textual passou pelo processo de *stemming* e lematização para redução do número de palavras do corpo textual usado na composição das *Word Embeddings*. Um dos desafios encontrados nos dados foi a ocorrência da mistura da língua nativa, o alemão, com inglês em algumas descrições de tarefas e para mitigar esse problema utilizou-se um tradutor automático.

Um método para desenvolver o modelo utilizado na pesquisa de Dan *et al.* (2020) não é definido. Entretanto, é apresentada a arquitetura base do modelo e também uma segunda arquitetura que concatena o resultado do modelo base com as informações dos metadados para a realização da estimativa final.

Os resultados obtidos nos experimentos do trabalho de Dan *et al.* (2020) são considerados pelos autores como encorajadores. As métricas avaliadas foram a acurácia que ficou acima de 55% e a média da magnitude do erro relativo (MMRE) que ficou por volta de 35%. Segundo os autores, esse desempenho fica próximo das estimativas realizadas por experiências. O modelo que concatenava a estimativa das descrições textuais com os metadados da tarefa teve acurácia e MMRE inferiores ao modelo base e sofreu com *overfitting*, dessa forma, os autores consideraram que essa abordagem necessitava de mais investigação.

3.5 Análise Entre os Trabalhos Relacionados e Esta Pesquisa

A pesquisa desenvolvida neste trabalho utiliza as técnicas que obtiveram melhores resultados nos trabalhos relacionados, assim como, algumas práticas que podem ser consideradas relevantes para as contribuições e também para validação dos experimentos.

Nesse sentido, uma forma interessante de fornecer o modelo de aprendizado de máquina desenvolvido e os demais dados que podem ser publicamente disponibilizados é a criação de um repositório público no site *GitHub*, uma prática que pode ser encontrada nos trabalhos de Tubelis (2018) e Choetkiertikul *et al.* (2019). Isso permite que outros trabalhos possam utilizar parte dos dados da pesquisa e também observar como a arquitetura do modelo de aprendizado de máquina foi implementada de maneira prática, possibilitando inclusive a reprodução dos experimentos para verificação dos resultados obtidos.

Em relação as técnicas de processamento de linguagem natural observa-se que a limpeza do texto para retirar fragmentos de códigos e outros caracteres que não fazem sentido para o texto assim como no trabalho de Tubelis (2018) e Dan *et al.* (2020) será uma das técnicas a serem reutilizadas nesse trabalho. O uso de *Word Embeddings* como técnica de representação de texto como vetor também surge como a opção mais viável ao se observar os trabalhos relacionados com os melhores resultados como o de Choetkiertikul *et al.* (2019).

Um dos elementos fundamentais da arquitetura do modelo a ser construído nesse trabalho trata-se da rede neural artificial BiLSTM. A opção por esse algoritmo de aprendizado de máquina se deu em razão da maioria dos trabalhos relacionados também o utilizarem.

O conjunto de dados a ser analisado neste trabalho será composto por repositórios de projetos *Open Source*. A construção desse conjunto de dados pretende ser realizada através da utilização dos algoritmos publicamente disponíveis por Tubelis (2018), onde é possível encontrar repositórios Jira públicos e obter os dados das tarefas dos projetos desses repositórios de forma

automática.

Apesar de que pode-se observar nos trabalhos relacionados a avaliação dos resultados obtidos pelos modelos de aprendizado de máquina com diversas métricas, encontra-se mais frequentemente a utilização das métricas MMRE e MAE. O presente trabalho tem a intenção de utilizar as métricas MAE e Erro Quadrático Médio (MSE) que serão apresentadas com mais detalhes na Seção 2.3.3.

No Quadro 2 encontra-se um resumo das principais características dos trabalhos relacionados citados neste capítulo e também os aspectos pertinentes a este trabalho.

Quadro 2 – Análise entre os trabalhos relacionados e este trabalho

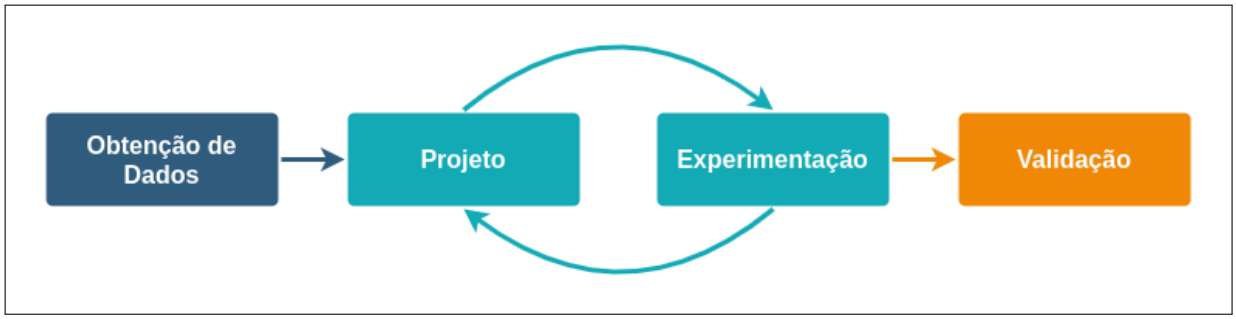
Trabalho	**Técnica de Representação de Palavras Como Vetor**	**Algorítimo Base**	**Composição do conjunto de dados**	**Entrada do Conjunto de Dados**	**Unidade de Esforço**	**Métricas de Avaliação**
(IONESCU *et al.*, 2017)	TF-IDF e doc2vec	SVR e GNB	Privado	Texto e Outros Atributos	Minutos	MMRE
(TUBELIS, 2018)	*Word Embeddings*	BiLSTM	Público e Privado	Texto	Horas	MAE
(CHOETKIERTIKUL *et al.*, 2019)	doc2vec	LSTM	Público	Texto	Pontos de Estória	MAE, MdAE e SA
(DAN *et al.*, 2020)	*Word Embeddings*	BiLSTM	Privado	Texto e Outros Atributos	Horas	MMRE
Este Trabalho	*Word Embeddings*	BiLSTM	Público	Texto	Minutos	MAE e MSE

Fonte: elaborado pelo autor.

4 MÉTODO DE PESQUISA

O método de pesquisa desenvolvido consiste em 4 fases principais: obtenção de dados, projeto, experimentação e validação. Esse método é o mesmo utilizado na pesquisa de Tubelis (2018). Essas fases possibilitam que se executem diversos experimentos durante a pesquisa. A Figura 7 ilustra as fases do método de pesquisa e como elas interagem entre si. Nas subseções seguintes, cada fase do método é apresentada.

Figura 7 – Fases do método de pesquisa



Fonte: Adaptado de (TUBELIS, 2018)

4.1 Obtenção de dados

Nessa fase, pretende-se utilizar os algoritmos em Python publicamente disponibilizados por Tubelis (2018) para encontrar repositórios Jira públicos de projetos *open source*. Após a descoberta dos repositórios, é possível fazer o armazenamento das tarefas com tempo de execução reportados.

4.2 Projeto

A fase de projeto consiste em atualizar parâmetros dos experimentos que podem influenciar no desempenho do modelo que está sendo desenvolvido. Essa fase deve ser executada antes do processamento dos dados e treinamento do modelo, pois são alteradas características da arquitetura do modelo e também de alguns passos realizados no pré-processamento dos dados. Após uma execução de um experimento em que houve melhorias em relação a uma versão anterior, uma nova versão é gerada para que se guarde as informações do experimento atual e se possibilite a atualização dos parâmetros na nova versão em busca de novas melhorias.

4.3 Experimentação

A experimentação é a fase em que o código de um experimento é realmente executado, gerando saídas importantes para a avaliação do modelo. Uma execução poderá ser interrompida quando problemas forem encontrados no pré-processamento e treinamento do modelo, como por exemplo uma métrica de avaliação ruim do modelo por muito tempo, o que demonstra que o processo de aprendizagem durante o treinamento não está ocorrendo.

4.4 Validação

Na fase de validação, avalia-se o modelo desenvolvido nas métricas de avaliação pré-estabelecidas ao fazer previsões para o conjunto de teste de um dataset. O conjunto de teste consiste em 30% das entradas de dados do conjunto de validação. As métricas são calculadas se aplicando a equação para os dados reais dos valores de saída, que nesse caso é o tempo de desenvolvimento da tarefa, e os valores previstos pelo modelo.

5 CONSTRUÇÃO DO MODELO

Este capítulo apresenta as etapas de obtenção, limpeza e análise do conjunto de dados utilizado na construção do modelo. Além disso, também detalha qual a arquitetura da rede neural do modelo e como foi realizado o seu treinamento.

5.1 Obtenção dos Dados

O primeiro passo de execução dos algoritmos publicamente disponibilizados por (TUBELIS, 2018) para obtenção dos repositórios Jira de projetos *open source* resultou na elaboração de uma lista de 74 repositórios, disponível nos Apêndices A e B.

Com a lista de repositórios elaborada o próximo passo executado foi a busca por tarefas contidas nesses repositórios. Para tanto, foi necessário a configuração de uma interface de programação de aplicação (*Application Programming Interface* (API)) de um motor de busca em uma plataforma de computação em nuvem que fornecia uma chave para que o algoritmo se conectasse com a API. Uma vez concluída a busca em um repositório, as tarefas eram salvas em uma lista dentro de um único arquivo de valores separado por vírgula (*Comma Separated Values* (CSV)). Por exemplo, as tarefas do repositório de URL `adaptlearning.atlassian.net`, representado pela sigla ADP estavam contidas em um arquivo nomeado `ADP_lab_raw.csv`.

Acredita-se que o conjunto de dados obtido e disponibilizado publicamente em um repositório Git[1] significa uma contribuição para a pesquisa em esforço de estimativa de *software*. Principalmente no que diz respeito à superação da dificuldade mencionada na Subseção 2.1.1 Item 1, sobre a pouca quantidade de dados públicos para projetos de softwares concluídos.

5.2 Limpeza do Conjunto de Dados

Ainda utilizando os algoritmos de Tubelis (2018), foi possível realizar uma limpeza inicial nos dados obtidos, essa limpeza eliminava do texto das tarefas alguns caracteres insignificantes e pedaços de código-fonte. Após essa limpeza inicial as tarefas passaram a ser salvas em arquivos de notação de objeto *JavaScript* (*JavaScript Object Notation* (JSON)) recebendo inicialmente o nome do repositório, como por exemplo `ADP_lab_clean.json`.

Com a conclusão da limpeza inicial observou-se que alguns repositórios continham descrições de tarefas vazias, dessa forma dos 74 repositórios iniciais foram considerados 70 para

[1] Disponível em https://github.com/abiliocastro/seeml

o próximo passo do experimento executado em um ambiente computacional interativo conhecido como *Jupyter Notebook* (PÉREZ; GRANGER, 2007).

Todos os arquivos dos 70 repositórios foram carregados em uma única estrutura de dados denominada *Pandas Dataframe* que faz parte da biblioteca para ciência de dados *Pandas* (TEAM, 2022). Após o carregamento das tarefas em um único *Pandas Dataframe*, verificou-se que o total de tarefas do conjunto de dados era superior a 97 mil (TEAM, 2022).

Quadro 3 – Diferenças do texto de uma tarefa antes e depois das limpezas

ESTADO DA TAREFA	**TÍTULO**	**DESCRIÇÃO**
Antes da limpeza	Grunt task runner: dev task does not copy json changes to the build	The \'watch\' task that runs if you do {{$ grunt dev}} will watch for changes to config.json - but then doesnt́ copy config.json into the build when this file is changed & saved.\r\n\r\nTo replicate:\r\n* run {{$ grunt dev}}\r\n* make a change to src/course/config.json and save the file\r\n* observe that the command line says {{>>File "src\\course\\config.json"changed}} and that the jsonlint:src, check-json and copy:courseJson tasks are run\r\n* go to build/course, observe that config.json has not been updated
Pós limpeza inicial	grunt task runner dev task does not copy json changes to the build	the watch task that runs if you do grunt dev will watch for changes to config.json but then copy config.json into the build when this file is changed saved to replicate run grunt dev make a change to and save the file observe that the command line says file src course config.json changed and that the and copy:coursejson tasks are run go to observe that config.json has not been updated
Pós limpeza final	grunt task runner dev task copy json change build	watch task run grunt dev watch change config json copy config json build file changed saved replicate run grunt dev make change save file observe command line say file src course config json changed copy coursejson task run go observe config json updated

Fonte: elaborado pelo autor.

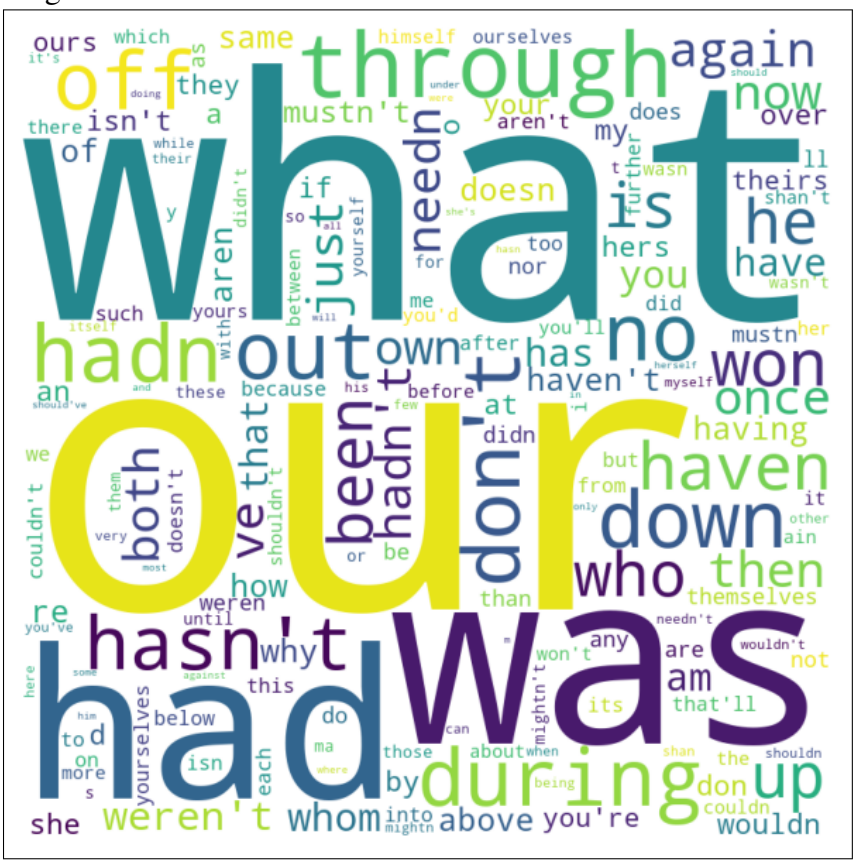
Continuando a análise dentro do *Jupyter Notebook* percebeu-se ainda a necessidade de uma limpeza final com os seguintes objetivos:

- Remoção de identificadores de tarefa: Os identificadores de tarefas no formato [IDENTIFICADOR] não contribuem com significado à tarefa e portanto foram removidos;
- Remoção de símbolos e pontuações: Alguns símbolos que não conseguiram ser

removidos na limpeza inicial como por exemplo "@", "$", "%", "*", "=", «"e sinais de pontuação como "!", "?", ",", "." foram então removidos nessa limpeza final;

- Remoção de palavras irrelevantes: Foi utilizado as (*stopwords*) da biblioteca NLTK para remoção de palavras irrelevantes na língua inglesa. A Figura 8 apresenta essas *stopwords* utilizadas (BIRD *et al.*, 2009);
- Lematização de palavras: Processo no qual se reduz uma palavra ao seu radical do dicionário. Por exemplo, as palavras trabalhando, trabalhou e trabalhava são reduzidas para a palavra trabalho (BERGMANIS; GOLDWATER, 2018);
- Remoção de números: Números isolados do contexto do título e descrição da tarefa também não acrescentavam significado e por essa razão foram removidos;
- Remoção de palavras curtas: Foi necessário remover algumas palavras com tamanho de apenas um caractere;
- Exclusão de entradas vazias: As linhas do conjunto de dados que possuíam título, descrição ou tempo de execução com valores nulos foram removidas.

Figura 8 – Palavras irrelevantes removidas do texto das tarefas

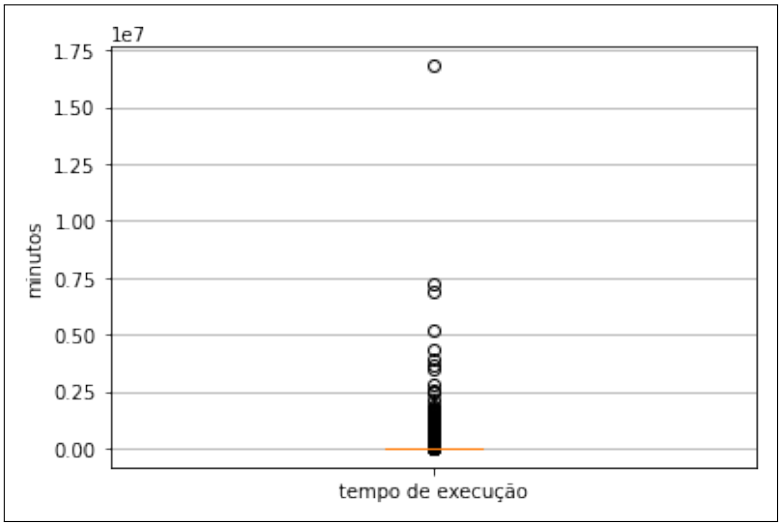


Fonte: elaborado pelo autor.

É possível observar como exemplo, no Quadro 3 as alterações que ocorrem no texto do título e da descrição de uma tarefa após cada processo de limpeza, onde é perceptível uma redução significativa no tamanho do texto após a limpeza final. O número de tarefas após a limpeza final foi reduzido para algo em torno de 72 mil tarefas.

A técnica de remoção de dados discrepantes conhecida em ciência de dados como remoção de *outliers* também foi utilizada para retirar do conjunto de dados as tarefas que possuíam tempo de execução com essa característica de fugir consideravelmente da distribuição normal. A Figura 9 trata-se de um diagrama de caixa para o tempo de execução das tarefas no conjunto de dados pós limpeza final, porém não é possível visualizar a caixa e os limites inferior e superior devido ao alto valor númerico dos *outliers*.

Figura 9 – Diagrama de caixa antes da remoção de *outliers*



Fonte: elaborado pelo autor.

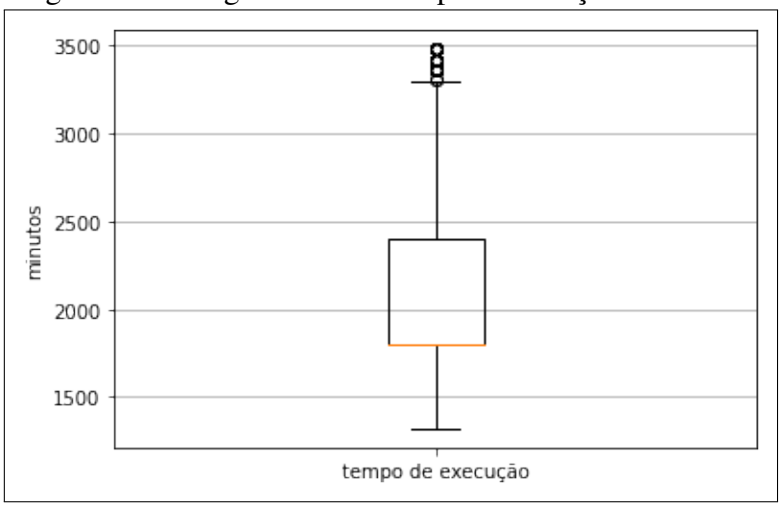
Uma vez realizada a remoção dos *outliers* é possível agora visualizar com clareza na Figura 10 o diagrama de caixa com todos os seus componentes: limites superior e inferior, quartis, a mediana na cor laranja e alguns poucos *outliers* acima do limite superior.

Com a remoção dos *outliers* o conjunto de dados teve seu número de entradas reduzido para aproximadamente 12 mil tarefas. A Figura 11 mostra como o número de entradas foi sendo reduzido após os processos de limpeza de dados.

5.3 Tradução de Tarefas

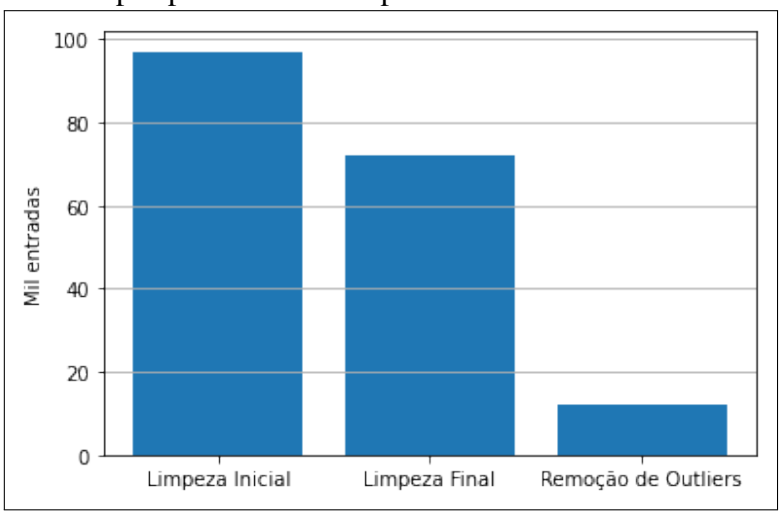
Uma questão que foi verificada em um dos experimentos e que resultou na adição de uma etapa de tradução de texto foi a ocorrência de tarefas com texto em português. Existia até

Figura 10 – Diagrama de caixa após a remoção dos *outliers*



Fonte: elaborado pelo autor.

Figura 11 – Redução do total de entradas do conjunto de dados após processos de limpeza



Fonte: elaborado pelo autor.

então, uma suposição errada de que todo o texto das tarefas obtidas já estavam em inglês.

Para garantir que todas as tarefas estavam em língua inglesa, foram criados dois métodos que utilizavam uma API especializada[2] em tradução de uma plataforma de computação em nuvem. O primeiro método realizava a detecção do idioma da tarefa e o segundo método realizava a tradução em si. O método de tradução chamava inicialmente o método de detecção de idioma e caso a tarefa estivesse em uma língua diferente do inglês era realizada então a tradução para o inglês.

Uma forma de evitar que a API de tradução fosse constantemente chamada foi salvar os dados traduzidos em um arquivo CSV para serem facilmente carregados novamente nos próximos experimentos a serem realizados. Para esse fim, foram utilizados os métodos `to_csv` e

[2] https://cloud.google.com/

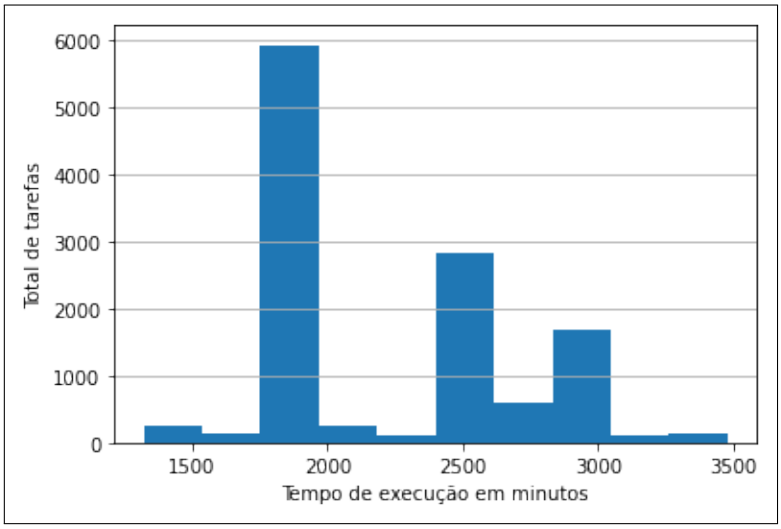
`read_csv` do objeto *Dataframe* da biblioteca *Pandas* para respectivamente salvar e ler os dados em CSV (TEAM, 2022).

5.4 Análise dos Dados

O conjunto de dados resultante após a limpeza e tradução das tarefas pode agora ter seus aspectos analisados para uma melhor compreensão de como as informações estão dispostas. Possibilitando assim, que algumas decisões importantes possam ser estabelecidas e também ocorra a validação de elementos essenciais na construção do modelo preditivo.

Uma primeira análise observada foi a distribuição de frequências dos tempos de execução das tarefas como pode ser visto na Figura 12 em que percebe-se uma grande concentração de tarefas entre 1500 e 2000 minutos. Outra análise realizada foi a estatística descritiva sobre a série de dados dos tempos de execução, essas informações estão apresentadas na Tabela 1.

Figura 12 – Histograma do tempo de execução das tarefas



Fonte: elaborado pelo autor.

Antes da criação do modelo de *Word Embeddings* foi necessário criar uma lista com todas as palavras das tarefas, incluindo título e descrição. Uma forma de visualizar as palavras mais frequentes, dessa lista de todas as palavras do texto, foi através de uma nuvem de palavras como se pode ver na Figura 13.

Percebe-se logo que na nuvem de palavras da Figura 13 existe uma predominância de termos relacionados à área de *software*, como *change*, *build*, *system*, *update* e entre outros. Algumas das palavras mais frequentes que podem ser observadas em destaque são *error*, *file*, *user*, *add*, *test* e *page*.

Tabela 1 – Estatística descritiva acerca dos tempos de execução das tarefas

INFORMAÇÃO	VALOR
Contagem de entradas	12041
Média	2186.46
Desvio Padrão	480.08
Valor Mínimo	1320
Percentis mais baixos (25%)	1800
Mediana (50%)	1800
Percentis mais altos (75%)	2400
Valor Máximo	3480

Fonte: elaborado pelo autor.

Figura 13 – Nuvem das palavras mais frequentes no texto

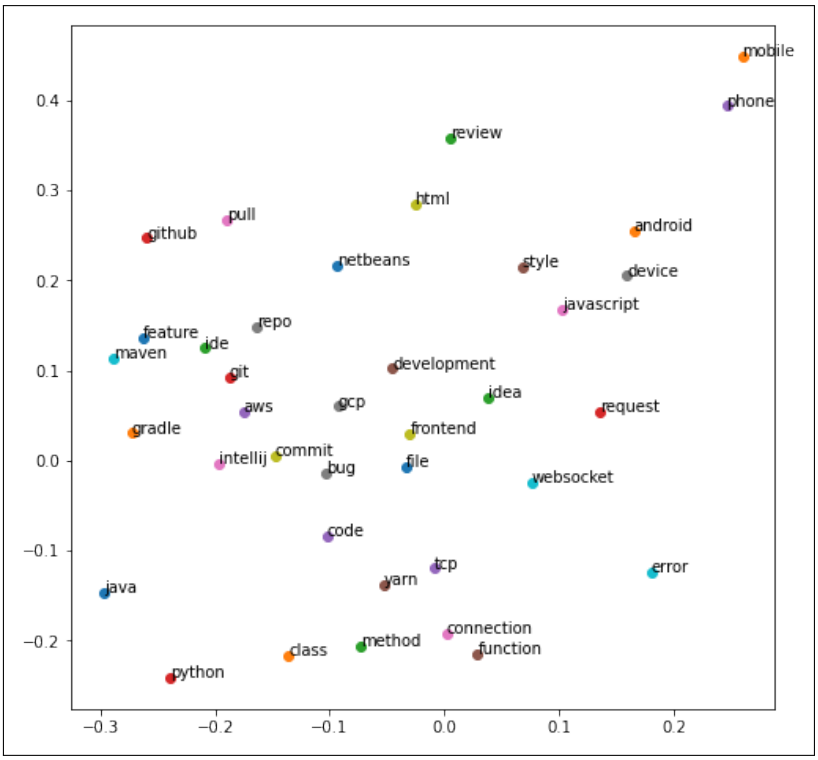
Fonte: elaborado pelo autor.

Embora a nuvem de palavras forneça uma visualização inicial das palavras mais frequentes do texto, após a criação do modelo de *Word Embeddings*, foram criados gráficos de barras para uma melhor compreensão do número total das palavras mais frequentes e também das palavras menos frequentes. Esses gráficos foram construídos agora considerando o vocabulário que o próprio modelo de *Word Embeddings* criou com o texto fornecido pelo título e descrição das tarefas.

Com o que pode ser observado no gráfico em barras horizontais do Apêndice C, contendo as 20 palavras mais frequentes do texto, confirma-se que as palavras citadas anteriormente como as que estavam em maior destaque na nuvem de palavras (*error*, *file*, *user* e *add*) realmente aparecem nas primeiras posições entre as palavras mais frequentes do texto. Enquanto que no Apêndice D podem ser observadas as 20 palavras menos frequentes no texto que nesse caso estão todas limitadas a 5 ocorrências, esse número corresponde ao número mínimo de contagem de frequência de palavras para serem consideradas como parte do modelo de *Word Embeddings*.

Outro aspecto importante que pode ser analisado no modelo são os vetores das palavras. Seria muito difícil obter uma visualização que relacionasse vetores de diferentes palavras em que todas as dimensões desses vetores, nesse caso 64, fossem utilizadas. Para obter uma visualização reduzindo esse número de dimensões para 2 se fez o uso do método de análise de componentes principais (*Principal Component Analysis* (PCA)) da biblioteca *Scikit-Learn* (PEDREGOSA *et al.*, 2011).

Figura 14 – Vetores de algumas palavras reduzidos para duas dimensões



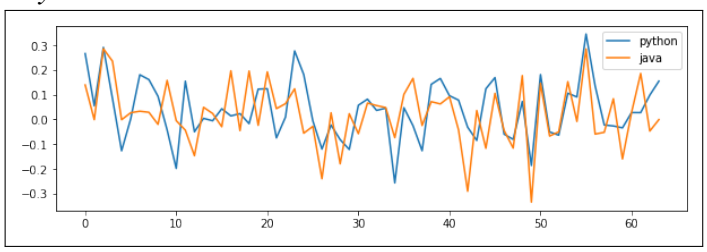
Fonte: elaborado pelo autor.

O gráfico da Figura 14 foi construído visando analisar a relação de algumas palavras que são comuns da área de desenvolvimento de *software*. É notável a proximidade de algumas linguagens de programação como *Java* e *Python* enquanto que a linguagens *JavaScript* e *HTML*,

que são mais voltadas para desenvolvimento de interfaces *web*, estão próximas entre si mas em um quadrante mais distante em relação à *Java* e *Python*. Alguns termos relacionados à redes de computadores como *TCP*, *Connection*, *Websocket* e *Request* estão relativamente próximos entre si. As palavras ligadas ao desenvolvimento de aplicações móveis como *Android*, *Device*, *Phone* e *Mobile* também estão agrupadas. Além desses agrupamentos de termos similares, verifica-se que termos mais genéricos se localizam mais ao centro do gráfico, como é o caso de *Frontend*, *File* e *Development*. Dessa forma, percebe-se que o modelo de *Word Embeddings* desenvolvido consegue definir bem o contexto em que cada vetor de palavra deve estar posicionado.

O modelo de *Word Embeddings* criado permite que se obtenha a similaridade entre duas palavras. Por exemplo, invocando-se o método `similarity` do objeto em que os vetores de palavras (`wv`) estão contidos dentro do modelo e passando como argumento as palavras *Python* e *Java* obtém-se como resultado algo em torno de 63% de similaridade. Pode-se também refazer essa análise de similaridade entre duas palavras na perspectiva de seus vetores, na Figura 15 percebe-se algumas semelhanças entre os vetores das palavras *Python* e *Java* com alguns picos bem próximos quando são da mesma polaridade.

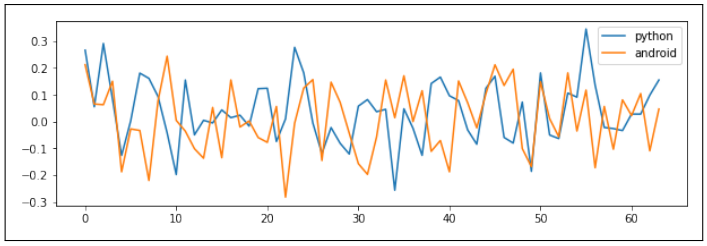
Figura 15 – Vetores de *Word Embeddings* das palavras *Python* e *Java*



Fonte: elaborado pelo autor.

Por outro lado, a similaridade entre as palavras *Python* e *Android*, por exemplo, é de apenas 12% ao se consultar o método `similarity` mencionado anteriormente. Enquanto que as discrepâncias de seus vetores podem ser verificados na Figura 16.

Figura 16 – Vetores de *Word Embeddings* das palavras *Python* e *Android*



Fonte: elaborado pelo autor.

5.5 Arquitetura Atual do Modelo Preditivo

A definição do modelo de *Word Embeddings* apresentado na Seção 5.4 representa uma etapa importante para que em seguida seja criado a matriz de *embeddings* que será fornecida como um dos parâmetros para criação de uma das camadas iniciais da rede neural do modelo preditivo.

Esta rede neural construída através da utilização dos componentes da biblioteca *Keras*, era composta de uma camada inicial de *Embeddings*, uma camada BiLSTM, uma camada de *Dropout*, uma camada *Dense*, mais uma camada de *Dropout* e por último uma camada *Dense* com um único neurônio para definir o valor final de saída. As camadas de *Dropout* são necessárias para evitar que o modelo perca generalização, um problema conhecido em aprendizado de máquina como *overfitting*, o valor de 80% de *Dropout* foi definido após vários experimentos como o melhor a ser utilizado (CHOLLET *et al.*, 2015).

O tamanho da entrada da camada inicial de *Embeddings* é definido pela média da quantidade de palavras dos documentos, nesse caso 33. O número de neurônios do LSTM é escolhido entre três valores: 16, 32 e 64. E o número de neurônios da primeira camada *Dense* é escolhido entre os valores: 32, 64 e 128. Para realizar a escolha do número de neurônios dessas duas camadas utilizou-se o *Keras Tuner* e o sintonizador *RandomSearch*, apesar da quantidade pequena de parâmetros a serem definidos, essa ferramenta propiciou a automatização do processo de escolha.

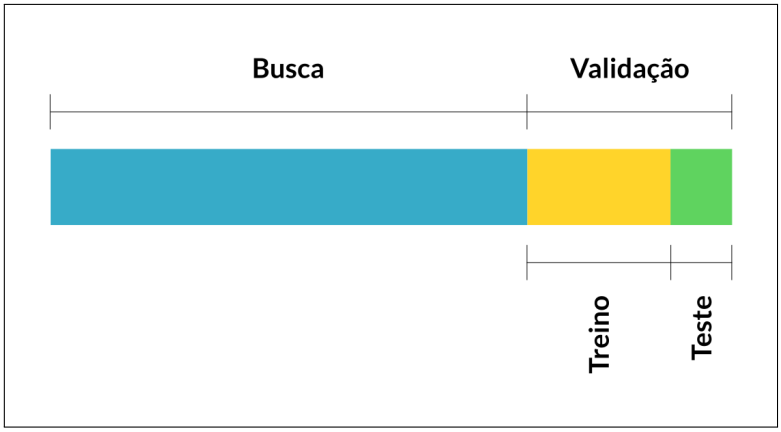
O conjunto de dados obtido foi particionado em 70% das instâncias para a busca no sintonizador *RandomSearch* mencionado anteriormente e os demais 30% para a validação do modelo. Ao se executar o sintonizador com a configuração de 20 épocas e 128 de tamanho de lote se obteve o resultado de 16 para a camada LSTM (ficando 32 unidades no total por fazer parte do BiLSTM) e 64 para a primeira camada *Dense*. O Apêndice E contém uma imagem onde pode-se visualizar a arquitetura final do modelo.

5.6 Treinamento do Modelo

Uma vez que a arquitetura do modelo foi definida se inicia então o seu treinamento. Os dados utilizados no treinamento correspondem a 70% do conjunto de validação. Esses dados formam o conjunto de treino que também é particionado durante o treinamento. Os outros 30% do conjunto de validação formam o conjunto de teste que será usado para avaliação das métricas.

A Figura 17 possibilita a visualização dessa divisão do conjunto de dados em um conjunto de busca e de validação, enquanto que o conjunto de validação é divido entre treino e teste.

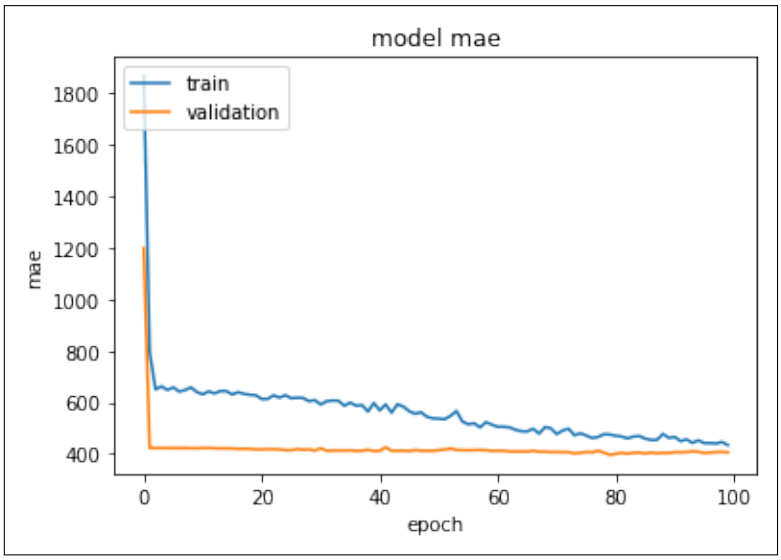
Figura 17 – Separação do conjunto de dados



Fonte: elaborado pelo autor.

O treinamento do modelo foi realizado com 100 épocas e com tamanho de lote de 128. A divisão do conjunto de treino para avaliação interna das métricas durante o treinamento foi de 30% para avaliação. Além disso, a métrica de perda definida anteriormente para o modelo foi a MSE. O histórico de treinamento do modelo foi salvo para que os gráficos das Figuras 18 e 19 fossem construídos.

Figura 18 – Evolução do treinamento do modelo para a métrica MAE

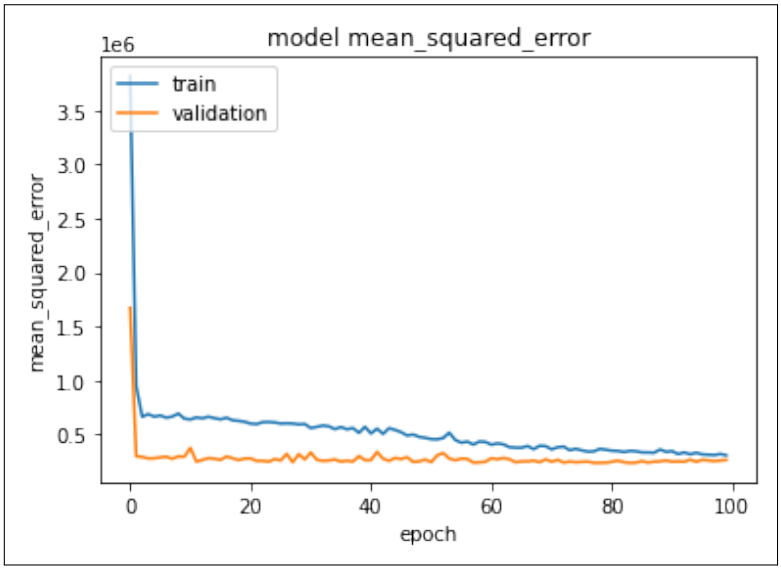


Fonte: elaborado pelo autor.

Estes gráficos apresentam a evolução do treinamento do modelo durante as épocas para as métricas MAE e MSE respectivamente. É perceptível em ambos os gráficos uma grande

queda das métricas de erro logo nas primeiras épocas de treinamento, porém essa queda continua ocorrendo mas de maneira mais lenta nas épocas seguintes fazendo com que a métrica de treino se aproxime da métrica de validação ao final do treinamento.

Figura 19 – Evolução do treinamento do modelo para a métrica MSE



Fonte: elaborado pelo autor.

6 RESULTADOS

Durante este trabalho vários experimentos foram realizados utilizando o método de pesquisa definido no Capítulo 4. Este capítulo pretende apresentar os resultados coletados de três experimentos executados com a versão final definida para o modelo e avaliar esses resultados com as métricas definidas na Subseção 2.3.3.

Utilizando-se dos dados de teste foram coletadas informações sobre três experimentos com três configurações diferentes do conjunto de dados. O uso de diferentes configurações do conjunto de dados é possível através da variação de um parâmetro chamado estado randômico `random_state` do método `train_test_split` que é responsável por fazer o particionamento dos dados, esse método também faz parte da biblioteca *Scikit-Learn* (PEDREGOSA *et al.*, 2011).

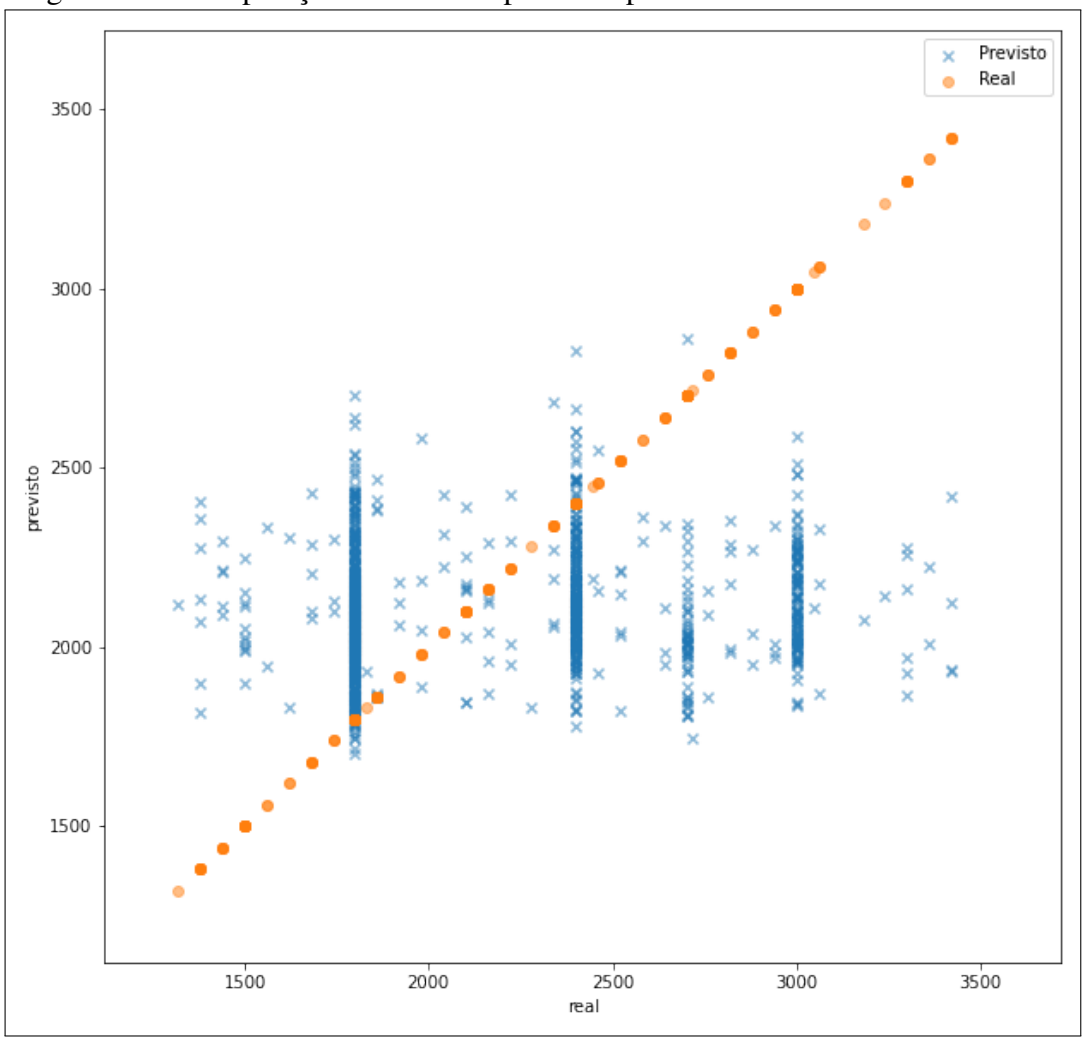
Com o modelo treinado realizam-se previsões utilizando o conjunto de teste e os resultados previstos podem ser comparados com o resultado real do tempo de execução da tarefa. Essa análise inicial pode ser feita utilizando-se uma visualização como a do gráfico da Figura 20. Observa-se assim a ocorrência de várias previsões distantes da previsão correta mas também algumas bem próximas e por essa razão é importante que se calcule o valor das métricas de avaliação MAE e RMSE para que assim tenha-se uma interpretação de quão grande pode ser o erro de previsão do modelo.

Dentro desse contexto, entende-se que é importante o estabelecimento de linhas base ou *baselines* de onde pode-se partir uma análise sobre a utilidade dos valores previstos pelo modelo. Dessa forma, foi definido o MAE e o RMSE dos valores Médio e Mediano do tempo de execução das tarefas no conjunto de treino. Usar essas *baselines* para a avaliação é como se fosse feito o questionamento: O modelo realmente realiza previsões significativas ou apenas usar a Média ou Mediana do conjunto de treinos é melhor? Essa estratégia também foi utilizada no estudo de Tubelis (2018) para validar os modelos desenvolvidos nos experimentos.

Os experimentos executados estão identificados como I, II e III. Para cada experimento verificou-se como as métricas se comportavam com a variação da frequência mínima de palavras que seria considerada para o modelo de *Word Embeddings*. Os valores utilizados nessa variação da frequência mínima foram: 2, 5, 10, 15 e 20. Procurando melhores visualizações dos resultados, decidiu-se pela separação dos gráficos de cada métrica para cada experimento.

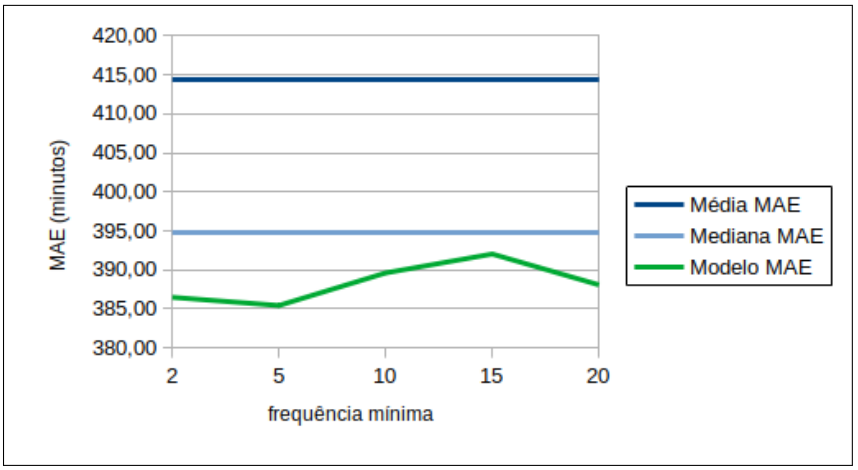
A Figura 21 apresenta os resultados do experimento I para a métricas MAE. Percebe-se que em todas as frequências mínima de palavras a métrica MAE do modelo possui um valor menor do que as *baselines* com destaque para o valor 5 que foi o menor entre todos.

Figura 20 – Comparação entre dados previstos pelo modelo e os dados reais



Fonte: elaborado pelo autor.

Figura 21 – Resultado da métrica MAE do experimento I

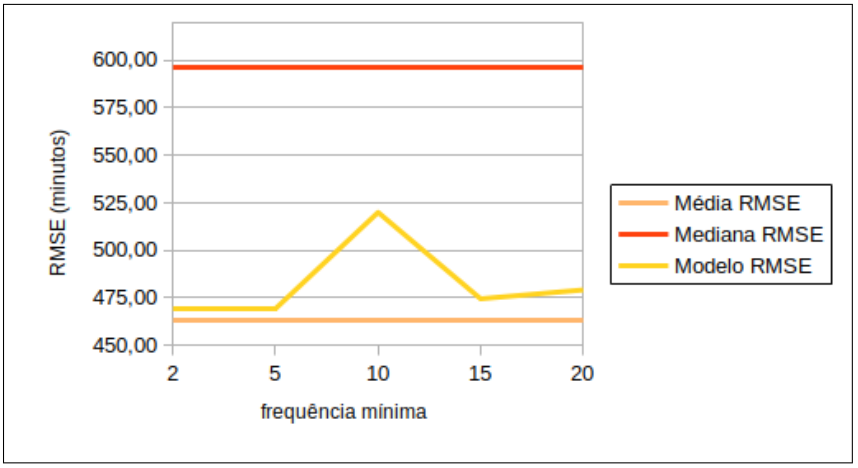


Fonte: elaborado pelo autor.

Entretanto, o resultado da métrica RMSE do experimento I que pode ser visualizado na Figura 22 apresenta um modelo com RMSE menor apenas que a *baseline* da mediana. Porém, deve-se ressaltar que somente para a frequência mínima de valor 10 que existe uma maior

distância entre o modelo e a *baseline* da média, para os outros valores de frequência mínima o modelo fica próximo mas ainda pior que a *baseline*.

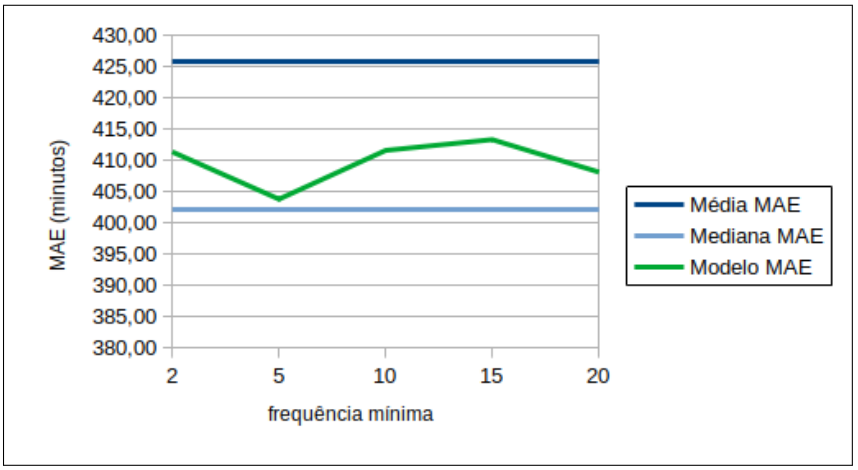
Figura 22 – Resultado da métrica RMSE do experimento I



Fonte: elaborado pelo autor.

Em relação ao experimento II a Figura 23 apresenta o resultado da métrica MAE em que o modelo foi melhor que a *baseline* da média. Em todos os demais valores de frequência o modelo foi pior que a *baseline* da mediana, apenas no valor de frequência mínima 5 que o modelo se aproximou um pouco da mediana.

Figura 23 – Resultado da métrica MAE do experimento II

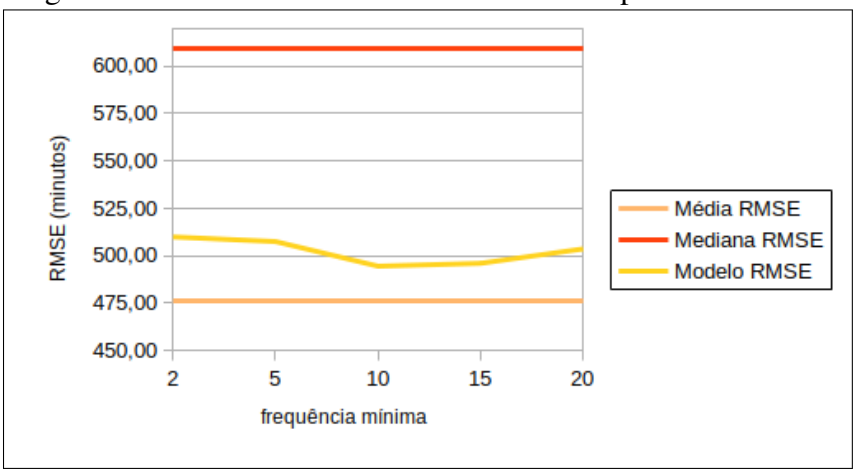


Fonte: elaborado pelo autor.

Ainda sobre o experimento II mas a respeito da métrica RMSE, como pode ser verificado na Figura 24, o modelo supera a *baseline* da mediana, porém não se aproxima tanto da *baseline* da média.

O experimento III tem seus resultados para a métrica MAE expostos na Figura 25 em que o modelo foi melhor que as duas *baselines* nos valores de frequência mínima 5, 10 e 15.

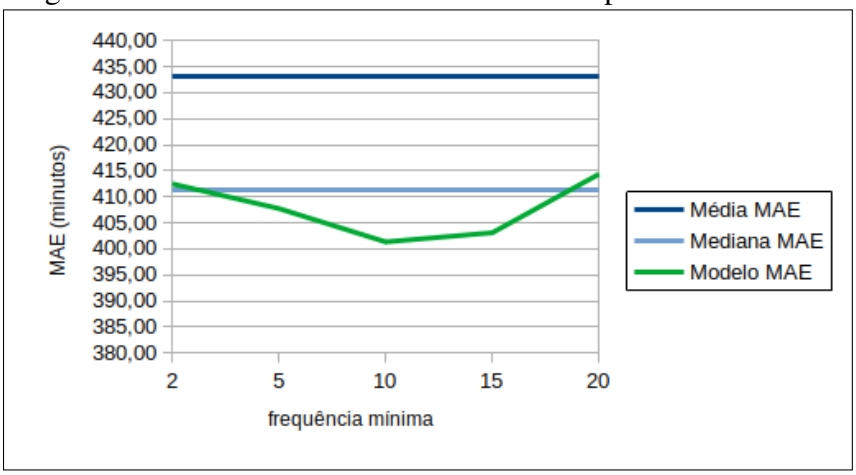
Figura 24 – Resultado da métrica RMSE do experimento II



Fonte: elaborado pelo autor.

Sendo que foi pior que a *baseline* da mediana nos valores 2 e 20 mas ainda assim muito próxima da *baseline* em questão.

Figura 25 – Resultado da métrica MAE do experimento III



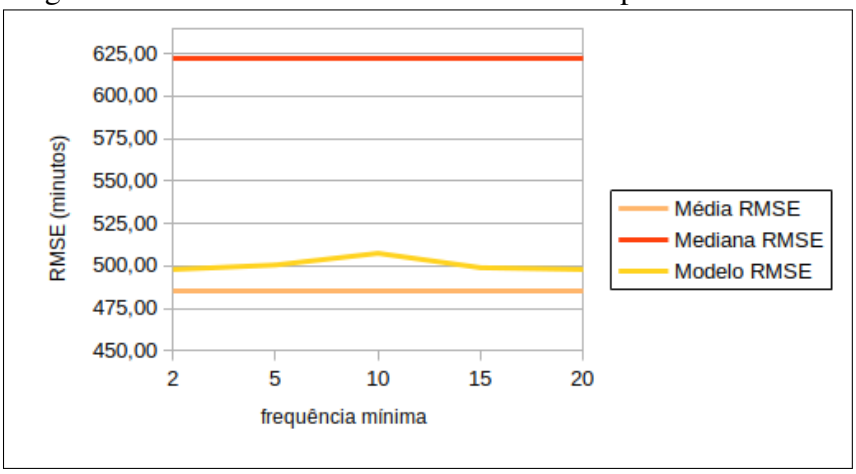
Fonte: elaborado pelo autor.

O resultado do experimento III para a métrica RMSE pode ser visualizado na Figura 26 e assim como nos dois experimentos anteriores o modelo foi melhor que a *baseline* da mediana. Porém nesse caso, para todos os valores de frequência mínima o modelo é pior mas ainda muito próximo da *baseline* da média.

Uma melhor verificação dos dados que foram expostos nos gráficos dos experimentos pode ser realizada com a análise do Apêndice F que contem todos os dados das métricas dos três experimentos e para todos os valores de frequência mínima. Essa tabela destaca em negrito os valores das métricas do modelo que foram melhor que ambas as *baselines*.

Em resumo, pode-se afirmar que o modelo apresentou métricas de erros que superam as *baselines* em dois experimentos. O menor valor de MAE obtido com o modelo foi de 386,49

Figura 26 – Resultado da métrica RMSE do experimento III



Fonte: elaborado pelo autor.

minutos o que corresponde a aproximadamente 6,44 horas enquanto que o menor valor de RMSE foi de 468,89 minutos que equivale a algo em torno de 7,81 horas.

7 CONSIDERAÇÕES FINAIS

O presente trabalho obteve dados de projetos *Open Source*, realizou a limpeza dos mesmos, analisou suas características e através do treinamento de *Word Embeddings* e redes neurais artificiais, com partes desses dados obtidos, construiu um modelo com o objetivo de prever o tempo estimado em minutos para a execução de tarefas de desenvolvimento de *software* através de sua respectiva descrição textual.

Os resultados dos experimentos realizados podem ser considerados significativos, tendo em vista que se assemelham com o de alguns trabalhos existentes na literatura como o de Tubelis (2018) que também realizou uma regressão com redes neurais a partir do texto das descrições de tarefas. Com isso, acredita-se que o objetivo principal da pesquisa foi alcançado.

O objetivo específico de construir uma base de dados de tarefas de projetos *Open Source* e disponibilizá-la publicamente também foi alcançado, tendo em vista que o conjunto de dados utilizado no trabalho está publicamente disponível em um repositório Git no site GitHub [1] que contém inclusive os *Jupyter Notebooks* dos vários experimentos realizados, possibilitando a verificabilidade da pesquisa com a reprodução dos experimentos.

A avaliação de diferentes arquiteturas do modelo, um dos objetivos específicos definidos, foi alcançado com sucesso durante os experimentos. Pois a avaliação das métricas esteve focada na escolha da melhor arquitetura para a RNA que foi evoluindo de uma arquitetura com muitas camadas até a arquitetura atual de 7 camadas em que os melhores parâmetros para duas dessas camadas foram definidos de forma automatizada com auxílio do *Keras Tuner*.

O último objetivo específico definido foi verificar a viabilidade do modelo desenvolvido no suporte à decisões de estimativa de esforço. Compreende-se que este objetivo específico foi alcançado pois o modelo em alguns experimentos obteve métricas de erro melhores que as *baselines* definidas. Isso representa uma possibilidade de criação de ferramentas para estimativa de esforço baseadas em modelos de aprendizado de máquina como o desenvolvido nessa pesquisa.

Os trabalhos futuros dessa pesquisa poderiam focar em questões como a construção de conjunto de dados de validação através de parcerias com empresas que desenvolvem *software*. Essa questão pode tratar-se de um desafio dependendo do contexto em que os futuros pesquisadores estejam inseridos, porque sabe-se que existem muitas razões legais envolvidas na utilização de dados de uma empresa privada, ainda que estes sejam apenas de uso interno. A construção e avaliação de modelos preditivos utilizando outros algoritmos de aprendizados

[1] Disponível em https://github.com/abiliocastro/seeml

de máquina seria uma opção interessante a ser observada. Uma sintonia de hiperparâmetros da rede neural artificial (*hyperparameter tuning*) que também poderia contribuir com a melhoria do modelo, pois na pesquisa atual apenas dois parâmetros que influenciavam no aprendizado da RNA foram otimizados. Outra questão que seria importante refere-se a avaliação das métricas do modelo caso fossem fornecidos mais dados sobre fatores direcionadores de custo além da descrição textual das tarefas conforme foi verificado em outros trabalhos disponíveis na literatura. Sugere-se ainda como trabalho futuro a aplicação de questionários ou algum outro método de pesquisa com gerentes de projetos de *software* visando entender o que esses profissionais, que lidam diretamente com estimativas, acreditam ser viável como ferramenta de estimativa de esforço apoiada por inteligência artificial e também se os menores valores das métricas de erros obtidas pelo modelo (6,44 horas para a métrica MAE e 7,81 para a métrica RMSE) correspondem à realidade dos erros de estimativa vivenciada por esses profissionais na prática.

REFERÊNCIAS

ABBAS, S. A.; LIAO, X.; REHMAN, A. U.; AZAM, A.; ABDULLAH, M. Cost estimation: A survey of well-known historic cost estimation techniques. **Journal of Emerging Trends in Computing and Information Sciences**, v. 3, n. 4, p. 612–636, 2012.

BARDSIRI, A. K.; HASHEMI, S. M. Software effort estimation: a survey of well-known approaches. **International Journal of Computer Science Engineering (IJCSE)**, v. 3, n. 1, p. 46–50, 2014.

BERGMANIS, T.; GOLDWATER, S. Context sensitive neural lemmatization with Lematus. In: **Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)**. New Orleans, Louisiana: Association for Computational Linguistics, 2018. p. 1391–1400. Disponível em: https://aclanthology.org/N18-1126. Acesso em: 16 jan. 2022.

BIRD, S.; KLEIN, E.; LOPER, E. **Natural language processing with Python**: analyzing text with the natural language toolkit. [*S. l.*]: "O'Reilly Media, Inc.", 2009.

BORADE, J. G.; KHALKAR, V. R. Software project effort and cost estimation techniques. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 3, n. 8, 2013.

BRAGA, E. L. d. S. I. **Revisão sistemática dos fatores de impacto na estimativa de esforço em projetos de software**. [S.l: s.n], 2019.

CHING, T.; HIMMELSTEIN, D. S.; BEAULIEU-JONES, B. K.; KALININ, A. A.; DO, B. T.; WAY, G. P.; FERRERO, E.; AGAPOW, P.-M.; ZIETZ, M.; HOFFMAN, M. M. *et al.* Opportunities and obstacles for deep learning in biology and medicine. **Journal of The Royal Society Interface**, The Royal Society, v. 15, n. 141, p. 20170387, 2018.

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; PHAM, T. T. M.; GHOSE, A.; MENZIES, T. A deep learning model for estimating story points. **IEEE transactions on software engineering**, IEEE, v. 45, n. 7, p. 637–656, 2019.

CHOLLET, F. **Deep Learning with Python**. Manning publications co, 2018.

CHOLLET, F. *et al.* **Keras**. [S.l], 2015. Disponível em: https://keras.io. Acesso em: 10 abr. 2022.

CONTRATRES, F. **Similaridade entre títulos de produtos com Word2Vec**. 2018. Disponível em: https://medium.com/luizalabs/similaridade-entre-títulos-de-produtos-com-word2vec-5e26199862f0. Acesso em: 6 set. 2021.

CUI, Z.; KE, R.; PU, Z.; WANG, Y. Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction. **arXiv preprint arXiv:1801.02143**, [S.l], 2018.

DAN, I.; CĂTĂLIN, R.; OLIVER, O. An nlp approach to estimating effort in a work environment. In: IEEE. **2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)**. [*S. l.*], 2020. p. 1–6.

DAVE, V. S.; DUTTA, K. Neural network based models for software effort estimation: a review. **Artificial Intelligence Review**, Springer, v. 42, n. 2, p. 295–307, 2014.

FARIA, P.; MIRANDA, E. Expert judgment in software estimation during the bid phase of a project–an exploratory survey. In: IEEE. **2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement**. [*S. l.*], 2012. p. 126–131.

HARRISON, M. **Machine learning pocket reference**: working with structured data in python. [*S. l.*]: O'Reilly Media, 2019.

HEEMSTRA, F. J. Software cost estimation. **Information and software technology**, Elsevier, v. 34, n. 10, p. 627–639, 1992.

IONESCU, V.-S.; DEMIAN, H.; CZIBULA, I.-G. Natural language processing and machine learning methods for software development effort estimation. **Studies in Informatics and Control**, v. 26, n. 2, p. 219–228, 2017.

JØRGENSEN, M. What we do and don't know about software development effort estimation. **IEEE software**, IEEE, v. 31, n. 2, p. 37–40, 2014.

LEE, K.-F. **Inteligência artificial**. [*S. l.*]: Globo Livros, 2019.

LIDDY, E. D. **Natural language processing**. [S.l: s.n], 2001.

MATSON, J. E.; BARRETT, B. E.; MELLICHAMP, J. M. Software development cost estimation using function points. **IEEE Transactions on Software Engineering**, IEEE, v. 20, n. 4, p. 275–287, 1994.

MAXIM, B.; PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. [*S. l.*]: Porto Alegre:[sn], 2016.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PÉREZ, F.; GRANGER, B. E. IPython: a system for interactive scientific computing. **Computing in Science and Engineering**, IEEE Computer Society, v. 9, n. 3, p. 21–29, maio 2007. ISSN 1521-9615. Disponível em: https://ipython.org. Acesso em: 16 jan. 2022.

PHANNACHITTA, P.; MATSUMOTO, K. Model-based software effort estimation–a robust comparison of 14 algorithms widely used in the data science community. **International Journal Of Innovative Computing Information And Control**, v. 15, n. 2, p. 569–589, 2019.

PORRU, S.; MURGIA, A.; DEMEYER, S.; MARCHESI, M.; TONELLI, R. Estimating story points from issue reports. In: **Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering**. [*S. l.*: *s. n.*], 2016. p. 1–10.

RUSSELL, S. J.; NORVIG, P. **Inteligência artificial**. [*S. l.*]: Elsevier, 2013.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of research and development**, IBM, v. 3, n. 3, p. 210–229, 1959.

SOMMERVILLE, I. **Engenharia de Software**. 9.. ed. São Paulo: Pearson Brasil, 2011.

TEAM, T. **Pandas-dev/pandas**: pandas. [S.l]: Zenodo, 2022. Disponível em: https://doi.org/10.5281/zenodo.7223478. Acesso em: 16 jan. 2022.

TIBCO. **What is a Neural Network?** 2021. Disponível em: https://www.tibco.com/reference-center/what-is-a-neural-network. Acesso em: 6 set. 2021.

TORFI, A.; SHIRVANI, R. A.; KENESHLOO, Y.; TAVAF, N.; FOX, E. A. Natural language processing advancements by deep learning: A survey. **arXiv preprint arXiv:2003.01200**, [S.l], 2020.

TUBELIS, M. **Estimating Software Development Task Effort Using Word Embeddings and Recurrent Neural Networks**. Dissertação (Mestrado) – NTNU, 2018.

USMAN, M.; BRITTO, R.; DAMM, L.-O.; BÖRSTLER, J. Effort estimation in large-scale software development: An industrial case study. **Information and Software technology**, Elsevier, v. 99, p. 21–40, 2018.

VERA, T.; OCHOA, S. F.; PEROVICH, D. **Understanding the Software Development Effort Estimation in Chilean Small Companies**. [S.l.:s.n], 2019.

APÊNDICE A – LISTA DE REPOSITÓRIOS

Quadro 4 – Lista de repositórios

SIGLA	REPOSITÓRIO
ADP	adaptlearning.atlassian.net
ALL	alluxio.atlassian.net
APE	apereo.atlassian.net
ART	artemis.atlassian.net
ATP	at.projects.genivi.org/jira
BRP	bugreports.qt.io
BOJ	bugs.openjdk.java.net
BFO	bugster.forgerock.org/jira
CAW	campingworld.atlassian.net
CSB	change.sos-berlin.com
CHA	cloud-haskell.atlassian.net
COL	colined.atlassian.net
CJX	cpcjax.atlassian.net
DFA	datafari.atlassian.net
DCM	dcm4che.atlassian.net
DAL	ddi-alliance.atlassian.net
DNZ	denizoguz.atlassian.net
DNT	dnntracker.atlassian.net
EVL	evolvevrn.atlassian.net
FBC	fbc-ss.atlassian.net
GEG	georgelewe.atlassian.net
IKA	ikasan.atlassian.net
INN	innovalog.atlassian.net
INS	inspectit-performance.atlassian.net
ISS	issues.apache.org/jira
ISF	issues.freepbx.org
ION	issues.onehippo.com
ISH	issues.shibboleth.net/jira
ISN	issues.sonatype.org
ITC	itcsas.atlassian.net
JEM	jembiprojects.jira.com
JIA	jira.atlassian.com
JAT	jira.automotivelinux.org
JEX	jira.exoplatform.org
JHY	jira.hyperledger.org
JLS	jira.lsstcorp.org
JMD	jira.mariadb.org

Fonte: elaborado pelo autor (2022).

APÊNDICE B – CONTINUAÇÃO DA LISTA DE REPOSITÓRIOS

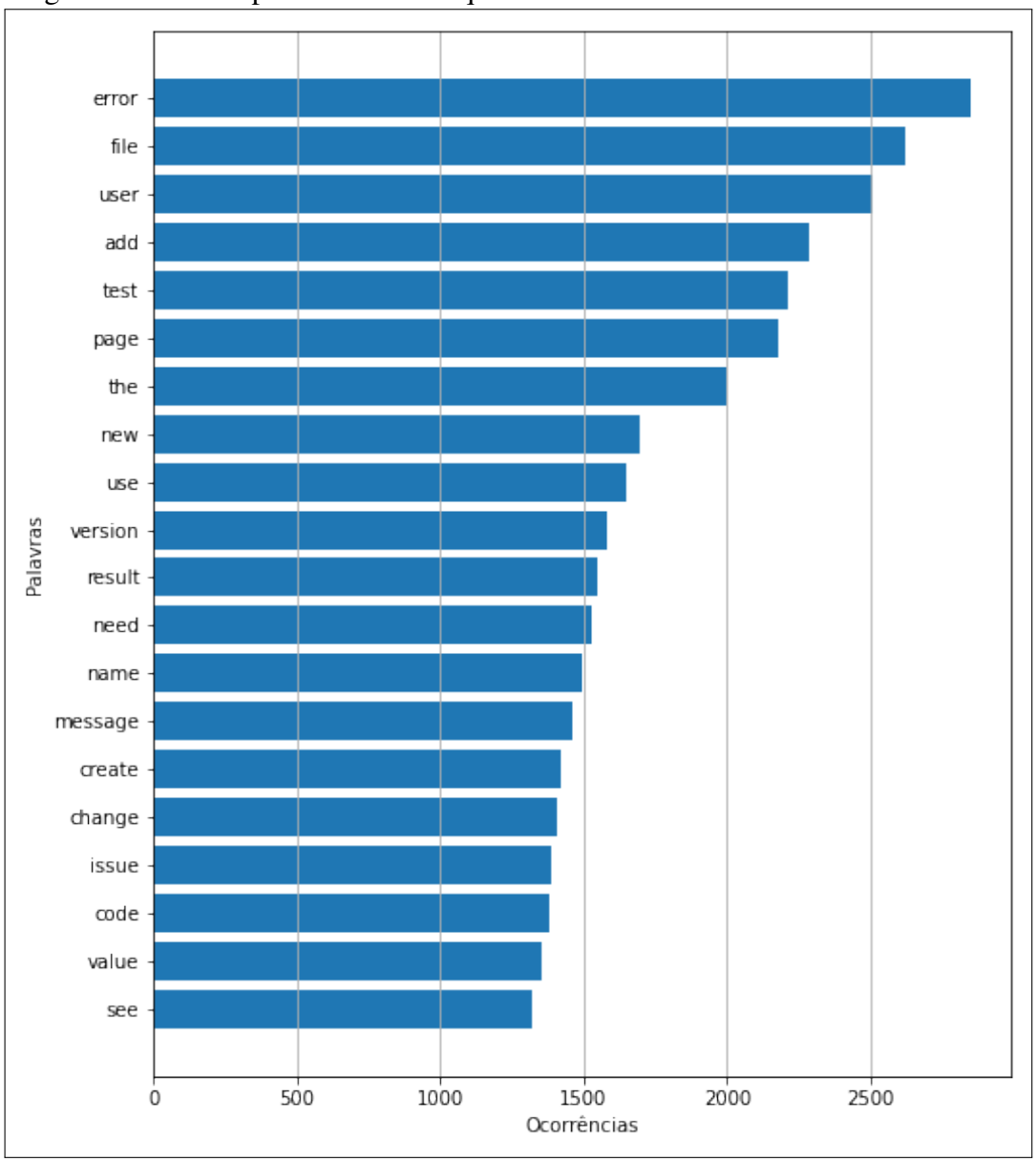
Quadro 5 – Continuação da lista de repositórios

SIGLA	REPOSITÓRIO
JMV	jira.mv.com.br
JON	jira.onap.org
JSA	jira.sakaiproject.org
JSE	jira.secondlife.com
JTO	jira.tools.ci.vodafone.com
KIT	kitchenup.atlassian.net
KIL	kylo-io.atlassian.net
LAK	lakeshore-learning.atlassian.net
LAM	lamestation.atlassian.net
LIB	librepilot.atlassian.net
LOG	logica.atlassian.net
MDB	mariadb.atlassian.net
MAT	matsim.atlassian.net
MET	metainf.atlassian.net
MIR	mirakel.atlassian.net
OAS	openandroidsvn.atlassian.net
ODX	openedx.atlassian.net
OHR	openehr.atlassian.net
OLE	openlibraryenvironment.atlassian.net
OSS	openoss.atlassian.net
OWI	openwips-ng.atlassian.net
PRE	presidecms.atlassian.net
PRI	prisepm.atlassian.net
PRO	project-fifo.atlassian.net
QCI	qcifltd.atlassian.net
QUA	qualityautomacao.atlassian.net
SDL	sdlc.corelogic.com
SER	servicedesk.wescef.com.au
SOF	softwareplant.com/jira
SOL	solerana.atlassian.net
TPP	thepluginpeople.atlassian.net
TIC	tickets.puppetlabs.com
TMO	tracker.moodle.org
TRI	trifolia.atlassian.net
USD	usdmo142.atlassian.net
WIS	wisoft.atlassian.net
XEN	xenoage.atlassian.net

Fonte: elaborado pelo autor (2022).

APÊNDICE C – AS 20 PALAVRAS MAIS FREQUENTES DO TEXTO

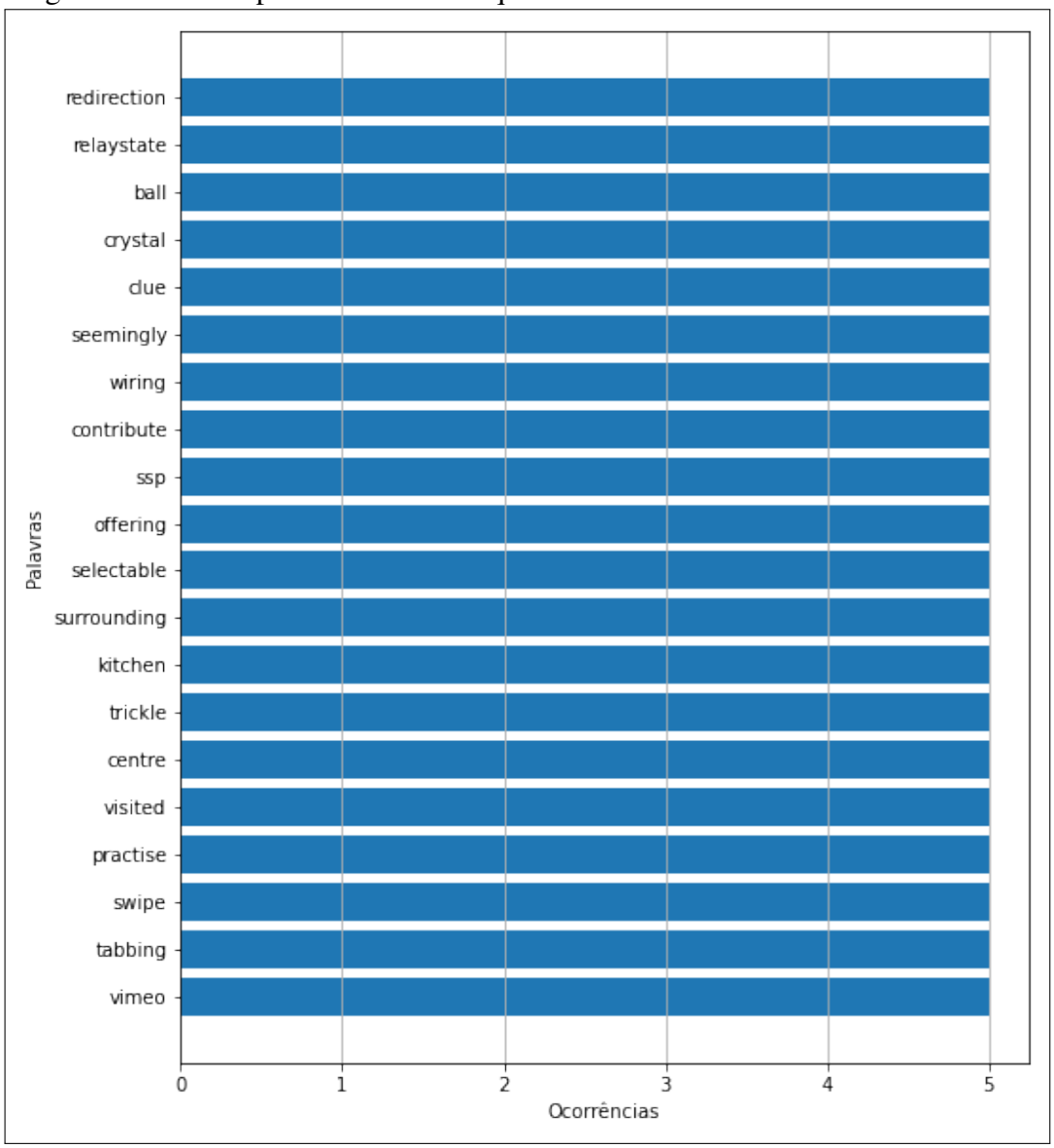
Figura 27 – As 20 palavras mais frequentes do texto



Fonte: elaborado pelo autor (2022).

APÊNDICE D – AS 20 PALAVRAS MENOS FREQUENTES DO TEXTO

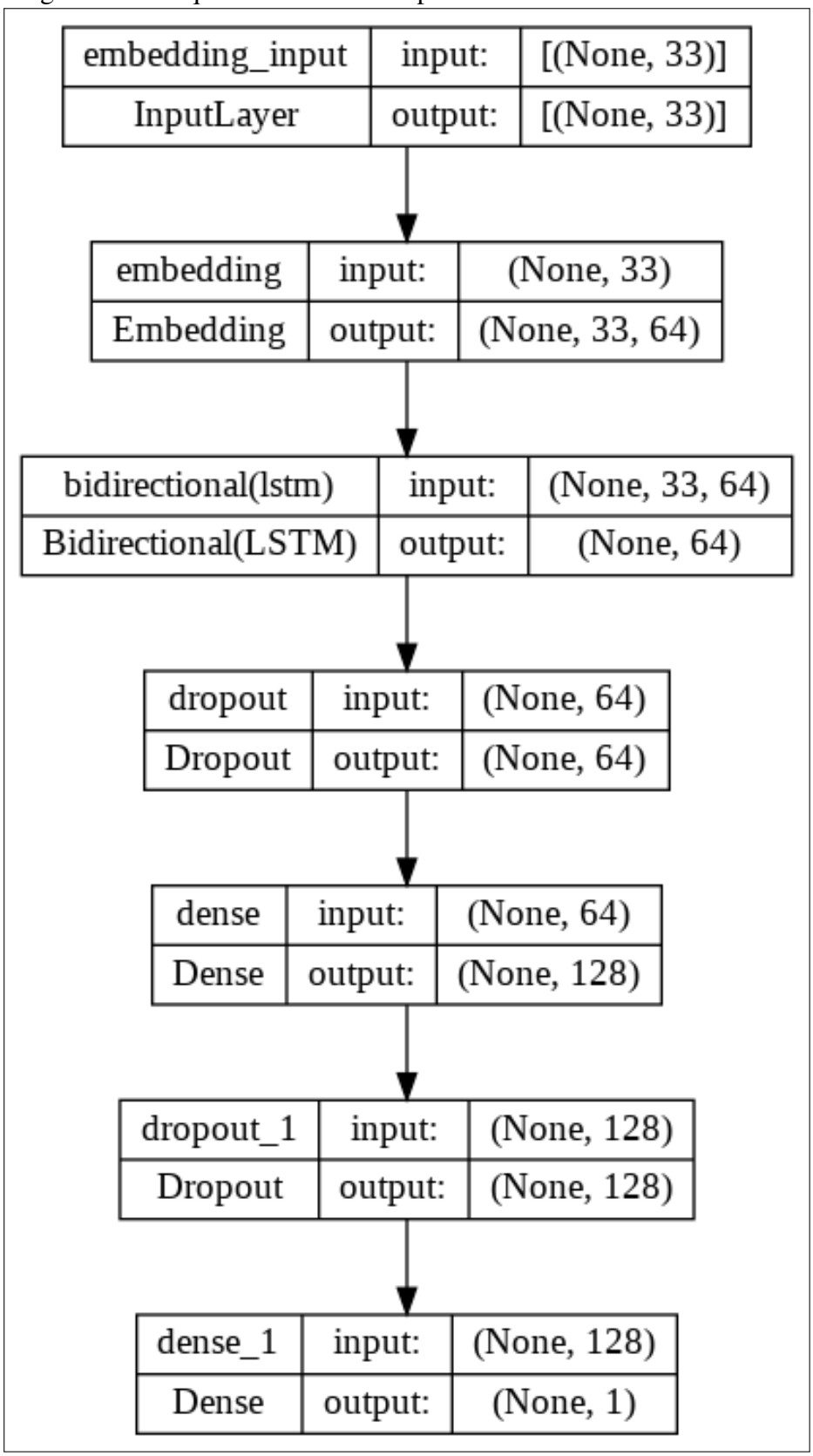
Figura 28 – As 20 palavras menos frequentes do texto



Fonte: elaborado pelo autor (2022).

APÊNDICE E – ARQUITETURA DO MODELO PREDITIVO

Figura 29 – Arquitetura do modelo preditivo



Fonte: elaborado pelo autor (2022).

APÊNDICE F – DADOS DAS MÉTRICAS DOS EXPERIMENTOS

Tabela 2 – Dados das métricas dos experimentos

			Frequência Mínima				
			2	**5**	**10**	**15**	**20**
Experimento I	MAE	**Média**	414,30	414,30	414,30	414,30	414,30
		Mediana	394,83	394,83	394,83	394,83	394,83
		Modelo	**386,49**	**385,44**	**389,61**	**392,03**	**388,08**
	RMSE	**Média**	462,95	462,95	462,95	462,95	462,95
		Mediana	596,22	596,22	596,22	596,22	596,22
		Modelo	468,89	469,06	520,05	474,56	479,21
Experimento II	MAE	**Média**	425,86	425,86	425,86	425,86	425,86
		Mediana	402,11	402,11	402,11	402,11	402,11
		Modelo	411,40	403,83	411,63	413,37	408,16
	RMSE	**Média**	476,04	476,04	476,04	476,04	476,04
		Mediana	609,28	609,28	609,28	609,28	609,28
		Modelo	510,01	507,54	494,56	496,07	503,68
Experimento III	MAE	**Média**	432,99	432,99	432,99	432,99	432,99
		Mediana	411,45	411,45	411,45	411,45	411,45
		Modelo	412,49	**407,75**	**401,33**	**403,11**	414,36
	RMSE	**Média**	484,97	484,97	484,97	484,97	484,97
		Mediana	622,50	622,50	622,50	622,50	622,50
		Modelo	498,19	500,48	507,40	498,92	498,10

Fonte: elaborado pelo autor (2022).