UNIVERSIDADE FEDERAL DO MARANHÃO

Programa de Pós-Graduação em Ciência da Computação

Weldson Amaral Corrêa

Aplicando Aprendizado de Máquina para Estimativa de Esforço no Desenvolvimento de Software

São Luís

2020

Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Aplicando Aprendizado de Máquina para Estimativa de Esforço no Desenvolvimento de Software

Weldson Amaral Corrêa

São Luís - MA
2020

Weldson Amaral Corrêa

Aplicando Aprendizado de Máquina para Estimativa de Esforço no Desenvolvimento de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFMA, como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Maranhão - UFMA

Centro de Ciências Exatas e Tecnologia - CCET

Programa de Pós-Graduação em Ciência da Computação - PPGCC

Orientador: Prof. Dr. Davi Viana dos Santos

Coorientador: Prof. Dr. Geraldo Braz Junior

São Luís - MA

2020

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Corrêa, Weldson Amaral.
Aplicando Aprendizado de Máquina para Estimativa de Esforço no Desenvolvimento de Software / Weldson Amaral Corrêa. - 2020.
66 p.

Coorientador(a): Geraldo Braz Junior.
Orientador(a): Davi Viana dos Santos.
Dissertação (Mestrado) - Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, São Luís, 2020.

1. Aprendizado de Máquina. 2. Aprendizado de Máquina Automatizado. 3. Esforço no Desenvolvimento de Software. 4. Estimativa de Esforço. I. Junior, Geraldo Braz. II. Santos, Davi Viana dos. III. Título.

Weldson Amaral Corrêa

Aplicando Aprendizado de Máquina para Estimativa de Esforço no Desenvolvimento de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFMA, como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Trabalho aprovado em São Luís - MA, 29 de Outubro de 2020:

Prof. Dr. Davi Viana dos Santos
Orientador

Prof. Dr. Geraldo Braz Junior
Co-Orientador

Prof. Dr. Luis Jorge Enrique Rivero Cabrejos
Universidade Federal do Maranhão

Prof. Dr. Valdemar Vicente Graciano Neto
Univeridade Federal de Goiás

São Luís - MA
2020

À minha família e amigos, em especial minha
e a todos que me ajudaram a chegar aqui.

Agradecimentos

Agradeço a Deus. A minha família, e a todos os meus amigos que sempre me incentivaram e apoiaram durante toda essa jornada.

Agradeço ao meu orientador Davi Viana o qual sempre me apoiou e me encorajou a não desistir deste trabalho. Sou grato pela paciência e experiência compartilhada durante esta jornada.

Agradeço ao professor e co-orientador Geraldo Braz Junior, o qual sempre foi uma inspiração. Agradeço também pelo incentivo para continuar na vida acadêmica e as lições que serão levadas para a vida. Meu muito obrigado por colaborar para o meu crescimento acadêmico, profissional e pessoal.

Agradeço em especial a Verianne dos Santos pelo companheirismo ao longo desta e de outras jornadas. Obrigado pelo incentivo e inspiração para buscar meus sonhos. Você é e sempre será importante na minha vida.

Agradeço aos professores Anselmo, Ari, Geraldo e João Dallyson por permitirem que fizesse parte do Núcleo de Computação Aplicada da UFMA, no qual aprendi a ser um melhor pesquisador e profissional. Agradeço também aos meus colegas de projeto Antônio Mourão, Alexandre, Victor Henrique, Mayara, Alan e Robert pela ajuda, conhecimentos trocados e experiências vividas.

Agradeço também a todos meus amigos, tanto os que estiveram me acompanhando diretamente neste processo, tanto os que me apoiaram indiretamente. Um agradecimento especial para Daniel, Artur, Dalai, Luiz Gonzaga, Pedro Júnior, David, Lucas e Eduardo. E para os que eventualmente não foram citados, me perdoem e meu muito obrigado.

A UFMA, CAPES pelos suportes estruturais e financeiros que fundamentais para o incentivo acadêmico e realização deste trabalho.

Por último, agradeço a quem estiver lendo pois só assim o conhecimento pode ser transmitido a outros.

"If I have seen futher it is by standing
on the sholders of Giants"
(Isaac Newton)

Resumo

Estimativas em projetos de software visam auxiliar profissionais na previsão de valores mais adequados para o desenvolvimento do sistema, impactando na qualidade do planejamento e execução das atividades do processo. Todavia, as organizações de software possuem dificuldades em realizar estimativas que representem uma aproximação mais adequada do esforço necessário para execução das atividades do projeto de software. Apesar da literatura apresentar técnicas para estimar o esforço, esta atividade continua não sendo trivial. Nos últimos anos, técnicas baseadas em algoritmos de Aprendizado de Máquina (AM) têm recebido destaque para auxiliar na resolução deste problema. Através de técnicas de AM, pode-se utilizar bases de dados de projetos já executados (*datasets*) para auxiliar em estimativas mais precisas. Entretanto, as estimativas geradas pelas técnicas de AM dependem do *dataset* que são aplicadas. Esta pesquisa tem objetivo de apresentar uma metodologia de estimativa de esforço usando algoritmo de aprendizado de máquina automatizado para ser capaz de generalizar as estimativas para diversos *datasets*. Para avaliar a metodologia proposta, foram conduzidos testes com 10 *datasets* e quatro métricas: *Mean Absolute Error*, *Median Magnitude Relative Error*, *Mean Magnitude Relative Error* e *Percentage Relative Error Deviation*. Os resultados demonstram que a metodologia proposta é consistente para estimativa de esforço em relação às métricas analisadas, indicando que a metodologia é promissora e consegue generalizar os resultados para múltiplos *datasets*.

Palavras-chaves: Estimativa de Esforço. Aprendizado de Máquina. Aprendizado de Máquina Automatizado. Esforço de Desenvolvimento de Software.

Abstract

Estimates in software projects aim to help practitioners predict more realistic values on software development, impacting the quality of software process activities regarding planning and execution. However, software companies have difficulties when carrying out estimations that represent adequately the real effort needed to execute the software project activities. Although, the literature presents techniques to estimate effort, this activity remains complex. Recently, Machine Learning (ML) techniques are been applied to solve this problem. Through ML techniques it is possible to use databases of finished projects (datasets) to help get more precisely estimations. However, the estimations depends on the dataset they are applied. This research propose a methodolody based on automatic machine learning to generalize the estimations through many datasets. To evaluate our methodology, we conducted tests with ten datasets using four metrics: Mean Absolute Error, Median Magnitude Relative Error, Mean Magnitude Relative Error and Percentage Relative Error Deviation. The results show that the proposed methodology has consistent estimations for sofware effort based on the employed metrics, which indicates that our methodology is promising and can generalize the results to other datasets.

Keywords: Effort Estimation. Machine Learning. Automatic Machine Learning. Software Effort Development.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFMA - Universidade Federal do Maranhão

AM - Aprendizado de Máquina

AutoML - *Automatic Machine Learning*

CASH - *Combined Algorithm Selection and Hyperparameter optimization*

COCOMO - *COnstructive COst MOdel*

TPOT - *Tree-Based Pipeline Optimization Tool*

RF - *Random Forest*

SVR - *Support Vector Regression*

GLM - *Generalized Linear Models*

SVM - *Support Vector Machine*

MLP - *Multi-Layer Perceptron*

MAE - *Mean Absoule Error*

MRE - *Magnitude Relative Error*

MMRE - *Mean Magnitude Relative Error*

MdMRE - *Median Magnitude Relative Error*

PRED - *Percentage Relative Error Deviation*

CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

Sumário

1 Introdução

1.1 Contexto

Um dos principais problemas de profissionais ligados ao desenvolvimento de software é estimar prazos de entrega e preço de projetos (CHARETTE, 2005; POSPIESZNY; CZARNACKA-CHROBOT; KOBYLINSKI, 2018). Estas variáveis de tempo e preço podem ser definidas a partir do esforço do projeto. O esforço para desenvolvimento de um projeto é o quanto de trabalho é necessário para desenvolvimento do software ou sistema.

A atividade de estimativa de esforço é o processo de predizer o quanto de esforço é necessário para o desenvolvimento de um software Song, Minku e Yao (2018), Wen et al. (2012). O esforço de desenvolvimento pode ser estimado no início e/ou durante o desenvolvimento do sistema (SONG; MINKU; YAO, 2018; CHARETTE, 2005). É importante para o desenvolvimento do software que as estimativas iniciais sejam o mais precisas possíveis, caso contrário, podem haver consequências indesejáveis para o projeto (CHARETTE, 2005). Subestimativas do esforço podem a levar a descumprimentos de prazos de entregas, ou ainda a um projeto inacabado Charette (2005). Por outro lado, superestimativas do esforço levam a desperdício de recursos, sobrepreço para o cliente, o que pode incorrer ainda na perda de contratos para empresas concorrentes (ABDELLATIF, 2018; CHARETTE, 2005).

A indústria de desenvolvimento de software vive uma expansão tanto no Brasil quanto no mundo. Segundo levantamentos de Software (2019), em 2018 o Brasil já contava com cerca de 5.294 empresas atuando no ramo de desenvolvimento de software. Esse mercado movimentou no Brasil cerca de US$ 10,6 milhões e globalmente mais de US$ 2 bilhões. Considerando as cifras movimentadas por este mercado e a importância que os softwares representam no dia-a-dia da população, todos os processos envolvidos no desenvolvimento de software são alvos de pesquisa para melhoria da qualidade, diminuição de custos e maior agilidade na entrega.

Considerando que este mercado esta cada vez mais globalizado, onde a concorrência tende a aumentar cada vez mais, estimativas de esforço precisas podem ser um diferencial para otimizar o emprego de recursos financeiros e de pessoal. Além disso, ao empregar prazos adequados para o desenvolvimento de um software pode ser evitado o surgimento de dívidas técnicas, uma vez que, a estimativa influência a qualidade de software (LIRA et al., 2015). Devido a importância da estimativa de esforço é compreensível o estudo de métodos e técnicas que possam auxiliar os profissionais da área a estimarem o esforço.

Os métodos de estimativa de esforço costumam ser classificados em três catego-

rias (MENDES; MOSLEY; WATSON, 2002): a) estimativa feita por especialistas, b) estimativa utilizando técnicas algorítmicas, como Pontos por Função, por exemplo, e c) técnicas de Aprendizado de Máquina (AM). Cada um desses métodos possuem características diferentes com vantagens e desvantagens. Nos últimos anos as técnicas de AM têm recebido maior destaque e gerando estimativas mais adequadas em relação as demais técnicas (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016). Essa melhoria pode ter ocorrido devido ao aumento de potência da IA gerado pelos avanços do processamento paralelo e de alto desempenho.

1.2 Definição do problema

Em projetos de desenvolvimento de software a maioria das decisões tomadas são baseadas em estimativas de esforço geradas no início do projeto (SATAPATHY; ACHARYA; RATH, 2016). Entretanto, no início do desenvolvimento existem poucos dados disponíveis para realização de estimativas coerentes. Adicionalmente, devem ser previstos os empecilhos como desistência de algum membro da equipe, falta de motivação, mudança de requisitos e diversos outros fatores que possam impactar a construção de um software (CHARETTE, 2005).

Estimativas realizadas por especialistas são subjetivas, e por mais experientes que sejam estes profissionais os mesmos estão sujeitos a erros, o que pode vir a prejudicar todo o andamento do projeto (WAZLAWICK, 2013). Em relação às técnicas de estimativa baseadas em modelos algorítmicos, por mais que possam ser auditáveis, estas ainda sofrem com a subjetividade. Estes modelos requerem como entrada algumas características subjetivas, por exemplo, o grau de conhecimento dos desenvolvedores em relação ao desenvolvimento orientado a objetos ou ainda a experiência dos desenvolvedores trabalhando em equipe (WAZLAWICK, 2013). Por último, as técnicas de AM têm sido utilizadas na estimativa de esforço, para auxiliar os engenheiros de software. A principal diferença em relação aos modelos algorítmicos consiste em estabelecer relação entre as características do projeto e o esforço de acordo com o conjunto de dados fornecido (MENDES; MOSLEY; WATSON, 2002).

Segundo Wen et al. (2012), Idri, Hosni e Abran (2016), as técnicas de AM têm obtido maior exito na tarefa de estimativa de esforço em relação às estimativas por especialistas e métodos algorítmicos. No entanto, segundo Idri, Hosni e Abran (2016) não existe um consenso sobre qual é a melhor técnica de AM para estimativa de esforço. Para usar técnicas de AM são necessários *datasets* com dados de projetos. Os resultados obtidos podem variar de um *dataset* para outro, ou seja, uma técnica que consegue um bom resultado num *dataset* A, obtém um resultado não satisfatório para o *dataset* B. De acordo com Wen et al. (2012) estas discrepâncias nos resultados podem ocorrer devido a

presença de *outliers*, valores ausentes, ou ainda, devido à adaptação a apenas um *dataset*.

Ainda segundo Idri, Hosni e Abran (2016), o problema de não haver uma técnica de AM que se destaque em relação as demais, levou pesquisadores a proporem a combinação de duas ou mais técnicas de AM, sob algum determinado critério para realizar as estimativas. Este conjunto ou combinação de técnicas é chamado de *ensemble*. Um *ensemble* também é uma técnica de AM. Com objetivo de diferenciar um *ensemble* das técnicas que o compõe, estas últimas são chamadas e técnicas de AM individuais (IDRI; HOSNI; ABRAN, 2016).

A principal vantagem de usar técnicas *ensemble* em relação às técnicas de AM individuais é a diversidade das técnicas que compõe o *ensemble* (CHANDRA; YAO, 2006). Duas técnicas são consideradas diversas se para um mesmo exemplo estas produzem erros diferentes (CHANDRA; YAO, 2006). Deste modo, ao combinar técnicas com uma boa diversidade num *ensemble*, as estimativas pobres de uma técnica de AM individual podem ser compensadas por estimativas mais adequadas de outras técnicas. Sendo assim, um *ensemble* deve produzir melhores estimativas que as técnicas que o compõem (SONG; MINKU; YAO, 2013).

Embora técnicas *ensemble* tenham melhorado as estimativas de esforço (IDRI; HOSNI; ABRAN, 2016), elas requerem um bom conhecimento por parte dos pesquisadores em relação a técnicas de AM para fazer boas combinações (JODPIMAI; SOPHATSATHIT; LURSINSAP, 2018). Além disso, os *ensembles* ainda não conseguem generalizar estimativas adequadas para os *datasets* (IDRI; HOSNI; ABRAN, 2016). Assim, é necessário ainda mais estudos e investigações sobre técnicas de AM para realizações estimativas de esforço.

Outro problema ao utilizar técnicas de AM, individuais ou *ensembles*, é encontrar os parâmetros mais adequados para estas técnicas (CHANDRA; YAO, 2006). Os parâmetros das técnicas de AM servem para ajustar estas ao contexto que está sendo aplicada (MARSLAND, 2015). Logo, é necessária experiência do pesquisador com ajuste de parâmetros.

1.3 Motivação e Justificativa

Atualmente as empresas possuem uma quantidade massiva de dados armazenados (BALAJI; ALLEN, 2018). No entanto, é necessária mão de obra especializada para extrair conhecimento a partir destes dados por existirem um vasto conjunto de possibilidades de decisão na escolha e configuração das técnicas de AM. Devido a esses fatores, o AutoML (*Automatic Machine Learning*) foi proposto como um algoritmo de otimização e aprendizado de máquina capaz de tomar decisões automaticamente e permitir produtividade (BALAJI; ALLEN, 2018). Atualmente as técnicas de AutoML estão sendo aplicadas a diversas áreas de pesquisa, na área de saúde (ALAA et al., 2019; ALAA; SCHAAR, 2018), internet das coisas (Chung et al., 2017), engenharia de software (TANAKA; MONDEN;

YÜCEL, 2019), entre outras.

Nesta dissertação, foram avaliadas duas técnicas de AutoML, Auto-Sklearn e TPOT, que são consideradas o estado da arte na área no momento da escrita deste texto (BALAJI; ALLEN, 2018). Estas técnicas podem facilitar o uso de técnicas de AM por pesquisadores e engenheiros de software, uma vez que necessitam de pouco conhecimento para serem utilizadas. Assim, ao propôr uma metodologia que utilize AutoML, permite-se que um profissional da indústria de software possa utilizar dados de projetos da empresa em que trabalhe para auxiliar a estimativa de esforço, possibilitando uma melhor alocação de recursos, evitando desperdícios e contribuindo para melhoria na qualidade do software.

A metodologia proposta deve ser capaz de ajustar-se a diferentes *datasets* para que seja possível a profissionais usar (IDRI; HOSNI; ABRAN, 2016). Considerando as características dos AutoML é possível ter êxito na resolução deste problema.

A metodologia apresentada neste projeto de pesquisa tem como premissa buscar solução para um dos grandes problemas da engenharia de software que é a estimativa de esforço. A solução proposta baseia-se no uso de técnicas AutoML, que são capazes de se ajustarem aos conjuntos de dados que são aplicados, permitindo assim, aplicar a metodologia em *datasets* locais e tendo auxílio na estimativa de esforço.

1.4 Objetivos

O principal objetivo desta pesquisa de mestrado consiste em analisar e desenvolver uma metodologia automatizada para estimativa de esforço no desenvolvimento de software e que seja capaz de generalizar as estimativas a diversos *datasets*. Possibilitando o uso de técnicas de AM para realização de estimativas em diferentes realidades de desenvolvimento de software, para isso será usado técnicas de AutoML para realização das estimativas.

1. Criação de uma metodologia de estimativa de esforço com AutoML;
2. Experimento controlado usando diferentes *datasets* para avaliar as estimativas de esforço geradas.
3. Gerar estimativas satisfatórias para os *datasets* analisados.

1.5 Organização da Dissertação

O trabalho segue a seguinte organização: o Capítulo 2, apresenta os conceitos de estimativa de esforço, bem como, as técnicas de AM utilizadas nos experimentos realizados durante este trabalho.

Capítulo 3, traz alguns trabalhos relacionados a este, com a utilização de técnicas de AM na estimativa de esforço no desenvolvimento de software. O capítulo descreve-os brevemente focando em suas principais características e contribuições.

No Capítulo 4, detalha-se a metodologia proposta de estimativa de esforço deste trabalho, assim como a implementação dos experimentos realizados. Enquanto que o Capítulo 5 apresenta os experimentos, resultados, discussão e as principais ameaças a validade deste trabalho.

Por fim, o Capítulo 6 apresenta as principais contribuições desta pesquisa e oportunidades que devem ser investigadas como trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta conceitos relacionados a estimativa de esforço no desenvolvimento de projetos de software, categorias de estimativa de esforço, métodos de estimativa de esforço e técnicas de aprendizado de máquina aplicadas para estimativa de esforço.

2.1 Estimativa de esforço

A estimativa de esforço no desenvolvimento de software diz respeito a quanto de esforço software é necessário para o desenvolver um software ou sistema (SONG; MINKU; YAO, 2018). A preocupação por parte de pesquisadores e profissionais da indústria de desenvolvimento de software por este tema vem desde os primórdios desta indústria. Segundo a literatura os primeiros métodos propostos para estimativa de esforço datam da década de 1980 (WEN et al., 2012) e desde então novos métodos têm sido criados e melhorados para resolução deste problema.

O problema da estimativa de esforço é inerentemente impreciso (DRAGICEVIC; CELAR; TURIC, 2017), em outras palavras, não há como se predizer com 100% de precisão o real esforço de um software. Isto ocorre porque o desenvolvimento de software é uma tarefa complexa, dado que este processo é dominado por interações entre pessoas (mesmo nos métodos ágeis) (WOHLIN et al., 2012). Assim, a estimativa de esforço no desenvolvimento de software torna-se uma tarefa igualmente complexa.

A estimativa de esforço de desenvolvimento de software considera muitas variáveis subjetivas. Há diversos exemplos destas variáveis, por exemplo, a familiaridade da equipe com o problema, experiência da equipe com programação, experiência da equipe com a linguagem ou *frameworks* utilizados, a experiência do próprio profissional que está estimando o esforço, motivação da equipe de desenvolvimento ao longo do projeto, dentre outras. Tendo em mente que todas estes variáveis subjetivas que usualmente são consideradas para estimar o esforço de um software, é possível perceber quão imprecisa essa tarefa é (DRAGICEVIC; CELAR; TURIC, 2017).

De acordo com Mendes (2014) os métodos de estimativa de esforço são divididos em três categorias, estimativa de esforço baseada em especialistas, modelos algorítmicos e estimativa baseada em AM. Cada uma das categorias apresentam diferentes métodos para estimativa de esforço com vantagens e desvantagens.

2.1.1 Estimativa de esforço por especialistas

A estimativa de esforço por especialistas é baseada na experiência do profissional em projetos anteriores. Este tipo de estimativa é totalmente subjetiva ao profissional que a faz e a precisão irá variar de acordo com a competência e experiência deste. A estimativa de esforço pode ser realizada por um ou mais especialistas como gerente de projetos e desenvolvedores (MENDES; MOSLEY; WATSON, 2002).

Estimativas de esforço baseadas na *expertise* de especialistas são úteis na ausência de dados empíricos. A principal desvantagem desta abordagem é a subjetividade, em outras palavras, a estimativa será tão boa quanto o parecer do perito e não há um meio de aferir a precisão até que a estimativa se prove errada. É importante ressaltar que anos de experiência não necessariamente significam nível elevado de competência. Além disso, mesmo especialistas competentes comentem erros ao fazerem estimativas de esforço. Dentre as técnicas baseadas na *expertise* de especialistas existem as técnicas *Delphi* e *Planning Poker* que serão apresentadas brevemente a seguir.

A técnica **Delphi** foi desenvolvida pela Rand Corporation no final da década de 1940 para fazer estimativas sobre eventos futuros (DALKEY, 1967). Esta técnica é usada para que um grupo de indivíduos cheguem a um consenso sobre um determinado assunto. A técnica *Delphi* se mostra útil quando existe a necessidade fazer estimativas sem o auxílio de grandes volumes de dados empíricos (BOEHM; ABTS; CHULANI, 2000). Embora existam limitações, esta técnica pode servir de apoio a outras técnicas como a COCOMO 2.

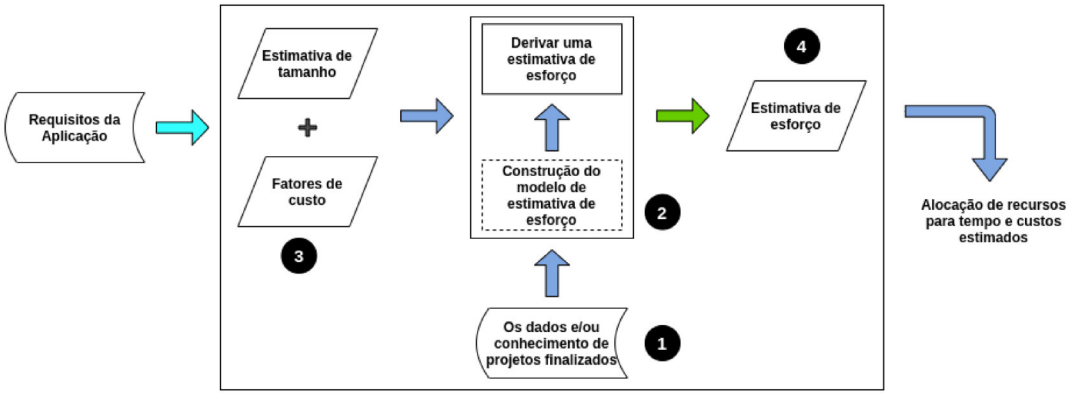
Planning Poker é outra técnica baseada na *expertise* de especialistas, sendo largamente utilizado e divulgado por metodologias ágeis (GRENNING, 2002). O Scrum, por exemplo, prega que todos os membros de um processo devem participar da atividade de estimativa, estes devem definir também um conjunto de atividades a serem realizadas num período de duas a quatro semanas após a reunião, este período é chamado de *Sprint*. A estimativa de esforço no *Planning Poker* baseia-se numa série numérica com objetivo de limitar o número de escolhas possíveis dos participantes, dando maior agilidade e diminuição no tempo gasto durante as estimativas.

As técnicas *Delphi* e *Planning Poker* oneram as equipes de desenvolvimento, uma vez que estas necessitam deslocar seus membros para reuniões de estimativa de esforço das atividades e projetos que devem ser desenvolvidos. Dentre as principais desvantagens das estimativas de esforço baseadas em *expertise* pode-se destacar a dependência do conhecimento humano, o que é prejudicial para estimativas de esforço se a rotatividade de pessoal for alta.

2.1.2 Modelos algorítmicos para estimativa de esforço

Os modelos algorítmicos utilizam-se de equações matemáticas para fazer a estimativa de esforço através de algumas variáveis que são dadas como entrada para estas equações. Estes modelos tentam estabelecer uma relação de dependência entre as características do projeto e o esforço requerido para o desenvolvimento (MENDES; MOSLEY; WATSON, 2002).

Figura 1 – Estimativa de esforço utilizando modelos algorítmicos



Fonte: Adaptado de (MENDES, 2014)

A Figura 1 apresenta um digrama onde (MENDES, 2014) mostra como os modelos algorítmicos são aplicados para estimativa de esforço.

1. Dados de projetos concluídos, são analisados para construção/geração de um modelo algorítmico;
2. Um modelo algorítmico é derivado a partir dos projetos que foram dados como entrada;
3. O modelo algorítmico recebe como entrada valores referentes às variáveis ou características do novo projeto;
4. Após a aplicação das equação é estimado o esforço do projeto.

Os *datasets* usados neste trabalho foram coletados seguindo alguns modelos algorítmicos, como COCOMO, Pontos de Função e Ponto de Caso de Uso.

2.1.2.1 COnstructive COst MOdel - COCOMO

O COCOMO que também é conhecido como COCOMO 81 foi um dos primeiros modelos algorítmicos propostos na literatura. Foi criado por Boehm em 1981 a partir de um estudo empírico (BARRY et al., 1981). Este modelo de estimativa de esforço é baseado

numa técnica conhecida como *Kilo Source of Line Codes* - KSLOC, em português, Linhas de Código em Milhares, consistindo em estimar o número de linhas de código que um programa terá. A estimativa de quantidade de linhas de código geralmente é feita por especialistas com base num histórico de projetos desenvolvidos. Os projetos analisados durante a criação do COCOMO tinham entre 2 e 100 KSLOC e foram escritos em diversas linguagens de programação.

Neste modelo algorítmico o esforço é calculado de acordo com a complexidade do projeto. São considerados três categorias de complexidade que levam em consideração o grau de informação do sistema a ser desenvolvido:

a) Implementação básica: quando a única informação disponível for o número de linhas estimado de linhas de código;

b) Implementação intermediária: se aplica quando os fatores relativos a produto, suporte computacional, pessoal e processo são conhecidos;

c) Implementação avançada: é usado quando o sistema deve ser dividido em subsistemas e as estimativas devem ser distribuídas por fase e atividade.

Outro fator que é levado em consideração é o tipo de projeto que deve ser desenvolvido. Para o cálculo do esforço são considerados novamente três categorias:

a) Modo orgânico: quando o desenvolvimento do software não envolve dispositivos de hardware e a equipe for experiente no desenvolvimento deste tipo de aplicação, em outras palavras, sistemas de baixo risco tecnológico e baixo risco de pessoal;

b) Modo semidestacado: se aplica a sistemas com maior grau de novidade para a equipe de desenvolvimento e que existam interações significativas com o hardware, mas, a equipe já possua algum conhecimento prévio, ou seja, sistemas em que a combinação de risco tecnológico e de pessoal for médio;

c) Modo embutido: é descrito como sistemas de alto grau de interação com diferentes tipos de dispositivos de hardware, ou que sejam embarcados e a equipe de desenvolvimento tenha dificuldade com a abordagem. Estes sistemas são considerados de alto risco tecnológico e/ou de pessoal.

Na **implementação básica** do COCOMO o esforço é calculado a partir da estimativa de KSLOC. O esforço (E) é calculado a partir da equação 2.1:

$$E = ab * KSLOC^{bb} \tag{2.1}$$

Tabela 1 – Tabela de valores de *ab*, *bb*, *cb* e *db* em função do tipo de projeto

Tipo de projeto	***ab***	***bb***	***cb***	***db***
Orgânico	2,4	1,05	2,5	0,38
Semidestacado	3,0	1,12	2,5	0,35
Embutido	3,6	1,2	2,5	0,32

onde *ab* e *bb* são obtidos a partir da Tabela 1.

Já o tempo (T) de recomendado para o desenvolvimento do software é dado pela equação 2.2:

$$T = cb * E^{db} \tag{2.2}$$

onde *cb* e *db* também são dados na Tabela 1.

A implementação básica do COCOMO é simples, mas, a estimativa gerada pelo modelo é limitada por considerar apenas a quantidade de linhas de código, enquanto que o esforço de um software depende de outros fatores que são considerados apenas no modelo intermediário.

O **modelo intermediário** do COCOMO considera outros aspectos do projeto além dos que são considerados no modelo básico. A Tabela 2 apresenta 15 fatores que são considerados no modelo intermediário.

Tabela 2 – Fatores de custos do modelo COCOMO intermediários.

Fatores influenciadores de custo	**Sigla**
Relativos ao produto	
Nível de confiabilidade requerida	RELY
Dimensão ba base de dados	DATA
Complexidade do produto	CPLX
Suporte computacional	
Restrições ao tempo de execução	TIME
Restrições ao espaço de armazenamento	STOR
Volatilidade da máquina virtual	VIRT
Tempo de resposta do computador	TURN
Pessoal	
Capacidade dos analistas	ACAP
Experiência no domínio da aplicação	AEXP
Capacidade dos programadores	PCAP
Experiência na utilização da máquina vitual	VEXP
Experiência na linguagem de programação	LEXP
Processo	
Adoção de boas práticas de programação	MODP
Uso de ferramentas atualizadas	TOOL
Histórico de projetos terminados no prazo	SCED

O modelo intermediário requer que para cada uma das características da Tabela 2 seja dado uma nota que varia de "muito baixo"a "extra alto". A partir da nota é dado um valor numérico correspondente. É possível notar que nem todas as notas são aplicáveis a todos os fatores.

Os valores numéricos dos fatores são combinados através de uma multiplicação, dado pela Equação 2.3:

$$EAF = RELY * DATA * CPLX * TIME * ... * SCED \tag{2.3}$$

A implementação intermediária do COCOMO estima o esforço a partir da equação 2.4:

$$E = ai * KSLOC^{bi} * EAF \tag{2.4}$$

onde os índices *ai* e *bi* são dados a partir da tabela Tabela 3

Tabela 3 – Tabela com valores de *ai* e *bi* de acordo com o tipo de projeto

Tipo de projeto	***ai***	***bi***
Orgânico	2,8	1,05
Semidestacado	3,0	1,12
Embutido	3,2	1,2

No modelo intermediário do COCOMO, os fatores influenciadores de custo são definidos de forma intuitiva introduzindo a subjetividade do aplicador deste método. A introdução de subjetividade faz com que a estimativa de esforço se torne tendenciosa. Neste trabalho foram empregados três *datasets* com dados históricos de projetos com esforço mensurado a partir do modelo intermediário do COCOMO, por isso é de fundamental importância entender o funcionamento deste modelo de estimativa de esforço.

Por fim, o COCOMO possui também uma **implementação avançada** que introduz outros conceitos que não são empregados nos dois modelos apresentados, como decomposição do projeto em subprojetos, assim como estimativas para cada fase do projeto. Como os *datasets* utilizados possuem dados apenas do modelo intermediário, não será apresentado como o modelo avançado do COCOMO funciona.

2.1.2.2 Pontos de Função

O método de Pontos de Função foi criado por Albrecht e Gaffney em 1983 (ALBRECHT; GAFFNEY, 1983) e assim como o COCOMO aplica equações para determinar o esforço necessário para o desenvolvimento de um software. Diferentemente do método COCOMO este método não é baseado na quantidade de linhas de código e sim, nos requisitos do sistema.

Uma vez que este método depende dos requisitos do sistema, tão logo os requisitos funcionais estejam definidos é possível fazer a aplicação do método de Pontos de Função. Para cada um dos requisitos são atribuídos valores numéricos, que, após o ajuste para a capacidade da equipe de desenvolvimento representará o valor de esforço para o desenvolvimento do sistema. Pode-se destacar também que este método independe da linguagem de programação e tecnologias envolvidas.

De forma geral para o desenvolvimento de um novo sistema devem ser considerados alguns fatores como:

a) Definição do escopo do sistema;

b) Identificação e atribuição de valor em pontos de função não ajustados para as transações sobre os dados, que são as entradas, consultas e saídas externas;

c) Identificação e atribuição de valor em pontos de função não ajustados(UFP) para dados estáticos, que são os arquivos internos e externos;

d) Definição do fator de ajuste técnico (VAF);

e) E por fim, calcular o número de pontos de função ajustados (AFP).

Para calcular o esforço neste método é necessário descobrir as funções do software. Aqui, funções são toda e qualquer transferência de informação para dentro ou fora do escopo do sistema e arquivo de dados acessíveis pelo usuário. Com base nessa definição os requisitos do sistema são interpretados e classificados como função. Nem todo requisito é uma função, assim como, há requisitos que possuem mais de uma função.

Após determinar as funções, é necessário classificá-las quanto a sua complexidade. Após esse passo, é possível obter o número de **Pontos de Função Não Ajustados - UFP** para o sistema. Esse número é calculado pela soma dos UFPs definidos para cada uma das funções do sistema. Os UFPs são definidos em função de fatores técnicos, o que presume uma linearidade no esforço, ou seja, quanto maior o UFP definido maior o esforço e maior o tempo de entrega. No entanto, como diferentes equipes produzem funcionalidades em ritmos diferentes, é necessário calcular também o fator técnico, que são os **Pontos de Função Ajustados - AFP**.

Os AFPs são considerados para obter-se valores mais realistas para o esforço. A técnica de pontos de função sugere 14 fatores de ajustes técnicos, conhecidos como *General Systems Characteristics - GSC*. Estes fatores recebem uma nota de zero a cinco. Zero indica que o fator não possui influência e cinco indica que o fator possui influência determinante no projeto. Os 14 fatores, são conhecidos *Value Adjustment Factor - VAF* e é calculado para o projeto completo e não para cada função individualmente.

Com os valores de UFP e VAF definidos é possível então calcular o AFP com a equação 2.5:

$$AFP = UFP * VAF \tag{2.5}$$

Uma vez determinado o AFP é possível determinar o esforço do projeto a multiplicando o AFP pelo Índice de Produtividade - IP da equipe, usando a equação 2.6:.

$$E = AFP * IP \tag{2.6}$$

O IP de uma equipe pode ser calculado a partir de um projeto da já desenvolvido cujo esforço seja conhecido. O esforço é contado como *desenvolvedor-mês*, ou seja, se 4 desenvolvedores trabalharam num projeto por 10 meses, o esforço foi de 40 desenvolvedor-mês. Se o último projeto de 800 AFP calculados, então o IP da equipe foi de IP = AFP/E = 800/40 = 20. O IP nesse exemplo é de 20 pontos de função ajustados por desenvolvedor-mês.

2.1.2.3 Pontos de Caso de Uso - PCU

O método de Pontos de Caso de Uso foi criado a partir da Tese de Gustav Karner em 1993 (KARNER, 1993). Este método como o nome sugere se baseia no diagrama UML de Caso de Uso de um software, e tão logo estes estejam definidos é possível aplicar esta técnica para fazer a estimativa de esforço.

De forma resumida o processo de aplicação deste método de estimativa de esforço consiste de seis partes:

a) Contar os atores e identificar sua complexidade;

b) Contar os casos de uso e identificar sua complexidade;

c) Calcular os PCUs não ajustados;

d) Determinar o fator de complexidade técnica;

e) Determinar o fator de complexidade ambiental;

f) Calcular os PCUs ajustados.

Neste método é considerado a **Complexidade de Atores - (*Unajusted Actor Weight*) UAW** que interagem com o sistema, atores são pessoas ou sistemas externos à aplicação com os quais a aplicação se comunica. Cada ator é contado uma única vez, mesmo este aparecendo múltiplos caso de uso. Além disso, atores não possuem subtipo, então caso existam especializações de um ator, é considerado apenas o mais genérico.

Após a análise dos atores, é verificado a complexidade dos Casos de Uso. Cada Caso de Uso é contado uma vez, e calculado sua complexidade que pode ser com base em três critérios, a quantidade de transações, movimentos de informação para dentro ou fora do sistema, ou ainda a quantidade média de classes necessárias para implementar o Caso de Uso. Por fim, a complexidade também pode ser calculada com base no risco de Caso de Uso. Ao final desta análise, a complexidade dos Casos de Uso são somados e é determinado o valor da **Complexidade dos Casos de Uso - (*Unadjusted Use Case Weight*) UUCW** da aplicação.

Após o cálculo da Complexidade dos Atores e Caso de Uso é possível encontrar o valor dos **Pontos de Caso de Uso Não Ajustados - (Unajusted Use Case Points) UUCP**, que é dado pela equação 2.7:

$$UUCP = UAW + UUCW \tag{2.7}$$

Após serem definidos os Pontos de Caso de Uso Não Ajustados é necessário fazer o ajuste dos pontos, para isso são usados dois critérios, Fatores Técnicos (relativos ao projeto) e Fatores Ambientais (relativos a equipe).

Os Fatores Técnicos ou *Technical Complexity Factor* - TCF são calculados com base na equação 2.8:

$$TCF = 0,6 + (0,01 * TFactor) \tag{2.8}$$

Onde, TFactor corresponde ao somatório de 13 fatores técnicos, por exemplo, complexidade de processamento, facilidade de uso, portabilidade, segurança, dentre outros, e cada um desses fatores recebe uma nota de 0 a 5.

Por outro lado, os **Fatores Ambientais ou *Environment Factor* - EF** são calculados com base em oito elementos sobre o ambiente de trabalho. Assim como no TCF, cada um dos fatores recebe uma nota entre 0 e 5, mas, com significado diferente dos TCF. Ao passo que no TCF é quantificado a influência de cada um dos fatores, aqui é analisado a qualidade destes fatores no ambiente de trabalho. Ao final da análise dos fatores, estes são somados e determinam o chamado *EFactor*, e o EF é calculado pela equação 2.9:

$$EF = 1,4 - (0,03 * EFactor) \tag{2.9}$$

Uma vez determinados os pesos dos atores e casos de uso, os fatores técnicos e ambientais, os pontos de caso de uso ajustados podem ser obtidos pela multiplicação destes

valores, conforme a equação 2.10:

$$UCP = UUCP * TCF * EF \tag{2.10}$$

Neste método o esforço pode ser finalmente calculado pela equação 2.11, onde os pontos de caso de uso é multiplicado pelo índice de produtividade da equipe.

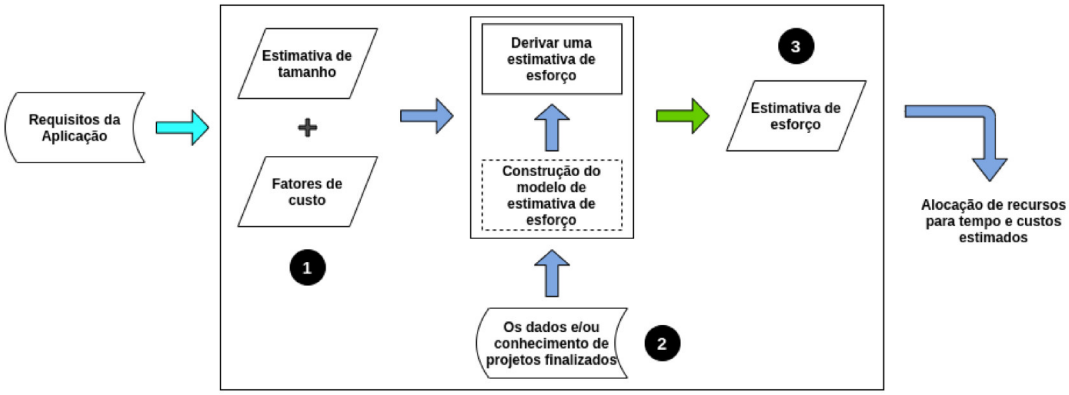
$$E = UCP * IP \tag{2.11}$$

Assim como no método de Pontos de Função, o índice de produtividade é calculado com base em projetos anteriores.

2.1.3 Técnicas de Aprendizado de Máquina

As técnicas de AM na estimativa de esforço começaram a ser aplicadas no início dos anos 90 (WEN et al., 2012). De forma geral são usados dados de projetos concluídos e a partir destes é estimado o esforço de um novo projeto. A Figura 2 apresenta um modelo genérico de como é aplicado técnicas de AM para a estimativa de esforço.

Figura 2 – Estimativa de esforço utilizando AM



Fonte: Adaptado de Mendes (2014)

1. Um conjunto de dados de projetos com seus respectivos esforços são dados como entrada apara o algoritmo de AM;
2. As características do novo projeto são passadas para o algoritmo de AM e este com base nos projetos que foram dados no passo anterior, calcula o esforço;
3. A técnica de AM informa o esforço estimado para o novo projeto.

Ao usar técnicas de AM para a estimativa do esforço, busca-se remover a subjetividade deste processo da construção de um software. Desde que sejam utilizados dados

apenas objetivos é possível remover completamente dados subjetivos, por exemplo, um estudo conduzido por Mendes, Mosley e Watson (2002) que utilizaram apenas dados objetivos para tentar fazer a estimativa do esforço. No entanto, como grande parte dos *datasets* coletados sobre a estimativa de esforço contém dados subjetivos estes ainda podem influenciar os resultados deste método.

Outra característica das técnicas de AM é que estas permitem a auditoria dos resultados, o que da maior transparência em relação a como o esforço foi estimado. De acordo com Wen et al. (2012) as estimativas realizadas por técnicas de AM têm obtido maior exito em relação aos demais. Por isso, neste trabalho escolhemos explorar este campo com objetivo de explorar algumas técnicas de *Automatic Machine Learning* (AutoML) na estimativa de esforço.

2.1.4 Métricas de estimativa de esforço

As métricas de estimativa de esforço servem para avaliar e permitir a comparação dos resultados de estimativas de esforço. Existem diferentes métricas de estimativa de esforço, no entanto, algumas das mais utilizadas dentro do escopo deste problema são as medidas de erro *Mean Magnitude Relative Error* (*MMRE*), *Percentage Relative Error Deviation* PRED(N), *Median Magnitude Relative Error* (*MdMRE*) e *Mean Absolute Error* (*MAE*) (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016; SONG; MINKU; YAO, 2018).

O *PRED(N)* representa a porcentagem de estimativas que estão dentro da margem de ±N em relação ao valor real de esforço. Na estimativa de esforço usualmente é utilizado o valor de 25% para N (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016), mostrando a porcentagem de estimativas que estão dentro do limite aceitável. Segundo (TIERNO; NUNES, 2013) dentro da literatura para PRED(25) um valor aceitável deve ser ≥75%, isto quer dizer que, pelo menos 75% das estimativas devem ter um erro percentual de 25% do valor real.

O erro relativo do inglês *Magnitude Relative Error* (*MRE*) indica o quanto o valor da estimativa encontra-se percentualmente longe do valor real. O valor de *MRE* pode ser calculado pela Equação 2.12. A média dos erros *MRE* é expressado pela métrica *MMRE*. É válido observar que quanto maior o valor de *MMRE* maior o erro das estimativas realizadas. Segundo Tierno e Nunes (2013) é considerado aceitável pela literatura um erro *MMRE* de até 25%. A Equação 2.13 apresenta como é calculado o valor de *MMRE*:

$$MRE = |(y_t) - \widehat{y}_t / y_t| \tag{2.12}$$

$$MMRE = \frac{\sum_{t=1}^{n} |(y_t) - \widehat{y}_t / y_t|}{n} \times 100 \tag{2.13}$$

onde y_t é o valor real, $\hat{y}_t$ o valor estimado e n a quantidade de dados da população.

A métrica *MMRE* é sensível a superestimativas, ou seja, estimativas com valor predito acima do valor real podem ter um erro acima de 100%, enquanto subestimativas podem ter um erro máximo de 100% uma vez que o esforço não pode ser negativo (TIERNO; NUNES, 2013). Por este motivo, passou-se a utilizar métricas que são menos sensíveis a superestimações.

A métrica *Median Magnitude Realative Error* (MdMRE) corresponde a mediana de *MRE* dado pela Equação 2.14. Essa é uma métrica menos afetada por valores superestimados uma vez que considera o valor médio do conjunto e quanto menor o valor apresentado por está métrica melhor os resultados.

$$MdMRE = median(MRE) \tag{2.14}$$

Mean Abosule Error (MAE) é uma métrica que avalia o erro absoluto do valor estimado em relação ao valor real de esforço e é dada pela Equação 2.15. De acordo com Shepperd e MacDonell (2012) esta métrica não é afetada por subestimações, uma vez que é simétrica. Assim como as métricas *MMRE* e *MdMRE* quanto menor o valor apresentado por esta métrica melhor a estimativa.

$$MAE = \frac{\sum_{t=1}^{n} |(y_t) - \widehat{y_t}|}{n} \tag{2.15}$$

onde y_t é o valor real, $\hat{y}_t$ o valor estimado e n a quantidade de dados da população.

2.2 Aprendizado de Máquina

Muitos dados são capturados e armazenados diariamente ao redor do mundo (MARSLAND, 2015). Estes dados podem ser fotografias, músicas, compras, etc. Com uma quantidade massiva de dados é importante para pesquisadores e a indústria descobrir como extrair informações a partir destes. Com uma quantidade pequena de dados é possível a pesquisadores tentar extrair informações manualmente, mas, a medida que a quantidade de dados disponível para análise cresce torna-se cada vez mais difícil para que humanos consigam fazer a análise dos dados. A partir desse ponto utilizar uma máquina ou computador para processar e extrair informações desses dados é interessante uma vez que computadores podem processar dados muito mais rápido que humanos e sem se cansarem.

Aprendizado de Máquina é a capacidade de fazer o computador modificar ou adaptar suas ações (sejam para fazer predições ou controlar um robô) de maneira a torná-las mais precisas. Essa precisão é medida por quão bem as ações escolhidas refletem a resposta correta (MARSLAND, 2015). Para que seja possível o aprendizado faz-se necessário que seja dado como entrada um conjunto de dados ou *dataset* para que o computador possa aprender.

Existem diversas maneiras de extrair informações de uma base de dados. Por exemplo, encontrar semelhanças para grupo de clientes de uma rede de supermercados, ou ainda, a partir de um conjunto de imagens de animais identificar qual é o animal numa fotografia. Outra possibilidade consiste em prever o preço de uma casa com base no tamanho, localização e ano de construção. Estes problemas são tratados dentro do Aprendizado de Máquina. Existindo diferentes áreas para tratar cada um destes problemas, como Aprendizagem Supervisionada e Aprendizagem Não Supervisionada.

Na **Aprendizagem Supervisionada** é dado um *dataset* rotulado, isto é, o valor de saída correto já é conhecido. Neste caso se tem a ideia que existe uma relação entre os valores de entrada e saída. Assim, o objetivo da máquina é encontrar uma função que represente essa relação. Os problemas da Aprendizagem Supervisionada são ainda classificados em problemas de "regressão"e "classificação". De forma resumida, em problema de regressão espera-se prever resultados de uma saída contínua (MARSLAND, 2015). Já em problemas de classificação deseja-se mapear as variáveis de entrada para categorias distintas.

Dois exemplos de problemas de regressão e classificação podem ser: dado um *dataset* com dados sobre tamanho de casas e preços, e tentar prever o preço. Como o preço é uma variável contínua, assim, este problema se encaixa na categoria dos problemas de regressão. Já um exemplo de regressão pode ser descrito como dado um *dataset* com fotografias de homens e mulheres. Neste tenta-se prever se aquela foto é de um homem ou mulher. No último exemplo temos duas categorias distintas, portanto, é um problema de classificação.

O problema da estimativa de esforço no desenvolvimento de software pode ser definido tanto como um problema de regressão quanto classificação (WEN et al., 2012; BANIMUSTAFA, 2018). O primeiro caso representa o esforço como uma variável contínua, onde ele vai de 0 ao infinito. Já para que este problema seja definido como um problema de classificação é necessário primeiro definir categorias de esforço, por exemplo, pequeno, médio e grande. Estas categorias, no entanto, dependem do pesquisador que as estiver utilizando. Neste trabalho, o problema da estimativa de esforço foi considerado como um problema de regressão, onde deve ser estimado um valor para o esforço.

Outra abordagem existente é a **Aprendizagem Não Supervisionada** onde é os dados são relacionados entre si para construção de agrupamentos (MARSLAND, 2015).

Existem diversas técnicas de AM para problemas de regressão e classificação, algumas inclusive servem podem ser utilizadas para resolver ambos os problemas. Na estimativa de esforço, diversas técnicas de AM já foram utilizadas, por exemplo: *Support Vector Machine*, *Random Forest*, *XGBoost*, Redes Neurais entre outras (WEN et al., 2012; AMARAL et al., 2019). Segundo (IDRI; HOSNI; ABRAN, 2016), não existe um consenso sobre qual a melhor técnica de AM para a estimativa de esforço, algumas obtém resultados satisfatórios para um *dataset* enquanto noutros os resultados não são bons. Assim, começou-

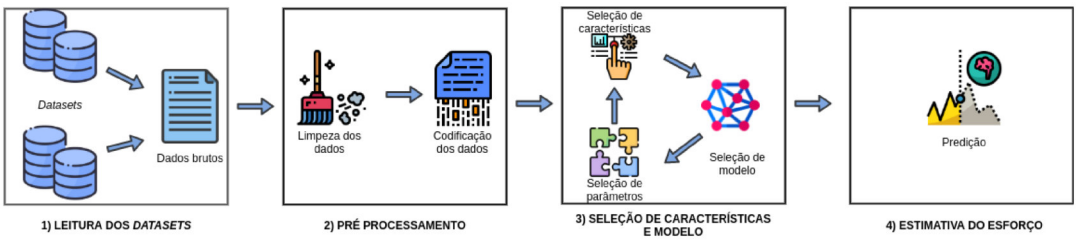
se a utilizar um *ensemble* de técnicas de AM, que nada mais é que uma técnica de AM que combina outras técnicas ou uma única técnica com diferentes parâmetros. A utilização de técnicas *ensemble* começou a ser empregada de modo a conseguir estimativas satisfatórias para o maior número de *datasets* com a mesma técnica de AM. (IDRI; HOSNI; ABRAN, 2016).

Ao utilizar uma técnica *ensemble* os pesquisadores ainda continuam com algumas preocupações, como encontrar os melhores parâmetros para a técnica utilizada, além de qual técnica utilizar. Com objetivo de solucionar alguns destes problemas, foi criado o *Automatic Machine Learning* ou AutoML que tem como finalidde encontrar quais os melhores parâmetros e técnicas para o *dataset* analisado.

2.2.1 *Automatic Machine Learning*

O *Automatic Machine Learning* também conhecido como AutoML, foi desenvolvido com objetivo de automatizar diversas partes da construção de um modelo de AM. A criação de um modelo de AM é um processo iterativo, demorado, e exaustivo. Esse processo envolve etapas de aplicação de variados algoritmos, além da análise dos melhores parâmetros para estes algoritmos. Os parâmetros das técnicas de AM fazem com que a técnica se adapte ao *dataset* que esta sendo estudado. Para tanto, é necessário que o pesquisador já possua um determinado nível de experiência para ajustar estes parâmetros. O AutoML, automatiza a etapa de encontrar parâmetros adequados e descobrir qual técnica de AM é mais adequada para o problema analisado. Assim, AutoML permite a pesquisadores automatizar diversas tarefas relativas à extração de conhecimento utilizando AM (BALAJI; ALLEN, 2018).

Figura 3 – Fluxo de uma estimativa de esforço utilizando AutoML



Fonte: Acervo do autor

A Figura 3 apresenta uma visão geral de como é realizada uma estimativa utilizando um AutoML. 1) Na leitura dos *datasets* é realizado a leitura do(s) *dataset(s)*, onde os dados já rotulados são dados como entrada. 2) Na Após etapa de pré-processamento são removidos dados que não são necessários, verificado a existência de dados com valores nulos ou incorretos e finalmente são codificados de forma a serem entendidos pelo algoritmo de AM. 3) Seleção de características e modelo, o AutoML aplica um conjunto de algoritmo de AM, assim como otimiza os parâmetros destes através de algum método de otimização

que depende da ferramenta de AutoML empregada e busca também quais os melhores parâmetros para a técnica de AM, ao final de todo esse processo é eleito a técnica de AM que obteve os resultados com melhor precisão. 4) A predição ou estimativa do valor esforço é realizada aplicando a técnica que obteve os melhores resultados na etapa anterior que foi a etapa de treinamento.

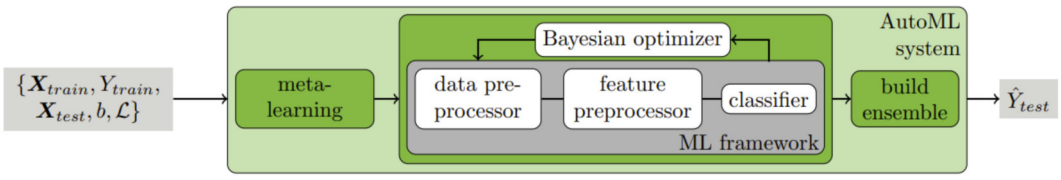
Neste trabalho a etapa de pré-processamento foi realizada manualmente e detalhada no Capítulo 4. Após o pré-processamento dos dados, é aplicado o AutoML para realização da estimativa de esforço. Na Figura 3 o AutoML representa a etapa 3, onde é realizado a busca por um modelo de AM e parâmetros para o *dataset* analisado.

2.2.1.1 Auto-Sklearn

O Auto-Sklearn foi criado baseado na definição de aprendizado de máquina automatizado formalizado por Thornton et al. (2013). Thornton et al. (2013) define o aprendizado de máquina automático como seleção de algoritmo combinado com otimização de hiperparâmetros, do inglês, CASH (*Combined Algorithm Selection and Hyperparameter optimization*). Dois importantes problemas são citados: (i) nenhum método de aprendizagem de máquina único funciona melhor em todos os conjuntos de dados, para cada bases de dados é necessário encontrar o método que mais se adequa (Sá; PAPPA, 2014) e (ii) alguns métodos de aprendizagem de máquina são muito dependentes da otimização dos seus hiperparâmetros. O Auto-Sklearn se propõe então a resolver os problemas (i) e (ii) como um problema único para resolver o problema CASH.

O Auto-Sklearn (FEURER et al., 2015) usa otimização Baysiana para a produção inteira de *pipelines* de AM, isto é, o pré-processamento dos atributos, seleção do algoritmo de AM e ajuste de hiperparâmetros. Buscando a combinação mais adequada entre o fluxo de AM e seus respectivos parâmetros. A Figura 4 apresenta como o Auto-Sklearn busca o modelo *ensemble* mais adequado para fazer estimativas.

Figura 4 – Processo de estimativa do Auto-Sklearn



Fonte: (FEURER et al., 2015)

O Auto-Sklearn foi desenvolvido usando algoritmos da biblioteca Sklearn. O espaço de busca para problemas de regressão é definido pelos seguintes algoritmos:

- *Adaboost*;

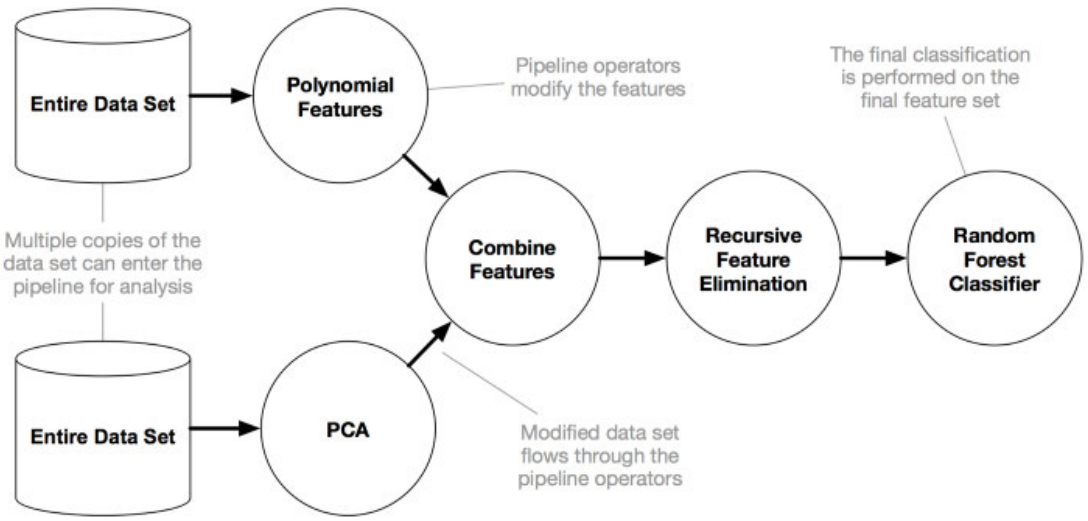
- *Decision Tree*;
- *Extra Trees*;
- *Gradiente Boosting*;
- *K-Nearest Neighbours*;
- *Random Forest*;
- *Xgradient Boosting*.

2.2.1.2 *Tree-based Pipeline Optimization Tool*

O TPOT (*Tree-Based Pipeline Optimization Tool*) (OLSON et al., 2016) tem como ideia principal criar inicialmente uma população inteira de *pipelines* de AM aleatórios e evoluí-los com mutações e cruzamentos de geração em geração. Para montar os *pipelines*, o TPOT usa vários algoritmos de seleção, construção e transformação de atributos e algoritmos de AM com ajuste de hiperparâmetros. Segundo Olson et al. (2016) é criado uma árvore utilizando operadores de *pipeline*, onde cada operador é responsável por adicionar, remover, transformar atributos ou dados, para que um operador final realizasse o processo de classificação ou regressão.

A Figura 5 apresenta o processo de construção de uma árvore de *pipeline*, onde cada círculo representa um operador.

Figura 5 – Fluxo de processamento do TPOT.



Fonte: (OLSON et al., 2016)

O TPOT cria várias cópias da base de treinamento para que os operadores possam combinar os atributos e dados modificados. Posteriormente, pode haver um processo de seleção de atributos para que enfim seja realizada a construção do modelo de estimativa.

O TPOT se baseia na biblioteca Sklearn para implementar seus próprios algoritmos. O espaço de busca para problemas de regressão é definido pelos seguintes algoritmos:

- *ElasticNetCV*;
- *ExtraTreesRegressor*;
- *GradientBoostingRegressor*;
- *AdaBoostRegressor*;
- *DecisionTreeRegressor*;
- *KNeighborsRegressor*;
- *LassoLarsCV*;
- *LinearSVR*;
- *RidgeCV*;
- *RandomForestRegressor*;
- *XGBRegressor*;
- *SGDRegressor*.

3 Trabalhos Relacionados

Este capítulo apresenta estudos que envolvem técnicas de AM aplicadas para estimativa de esforço no desenvolvimento de software. Os trabalhos retratados neste capítulo fazem a utilização de técnicas *ensemble* para a resolução deste problema. Além disso, é apresentado uma aplicação de AutoML dentro da Engenharia de Software. Os trabalhos apresentados neste capítulo são oriundos de um mapeamento sistemático da literatura.

3.1 Estimativa de Esforço com AM

A estimativa de esforço no desenvolvimento de software é um problema bem conhecido na literatura (WEN et al., 2012). Já foram propostas diversos métodos para fazer estimativas (MENDES; MOSLEY; WATSON, 2002), dentre os quais as técnicas de AM têm-se destacado (WEN et al., 2012). Os primeiros trabalhos que investigaram a aplicação de técnicas de AM, para a estimativa de esforço no desenvolvimento de software, remetem a década de 90 (WEN et al., 2012). Ao longo dos últimos anos, diversas técnicas de AM foram propostas, adaptadas ou criadas com objetivo de auxiliar a estimativa de esforço (IDRI; HOSNI; ABRAN, 2016), contudo, este é um problema que permanece aberto e atual (IDRI; HOSNI; ABRAN, 2016; SONG; MINKU; YAO, 2018).

Segundo Idri, Hosni e Abran (2016) apesar do grande número de estudos sobre estimativas de esforço com AM, não existe um consenso sobre qual melhor técnica de AM individual para estimativa de esforço no desenvolvimento de software. Shepperd e Kadoda (2001) sugerem ser mais adequado encontrar a melhor técnica AM individual para um contexto único, isto é, para um único *dataset*, que determinar a melhor técnica de AM para todas as estimativas de esforço. Esta dificuldade se dá, pela diferença de comportamento entre diferentes *datasets*. Wen et al. (2012), Pospieszny, Czarnacka-Chrobot e Kobylinski (2018) corroboram com a afirmativa que técnicas individuais produzem resultados discrepantes dependendo do *dataset* que foram aplicadas, segundo eles, essas diferenças acontecem por conta de *outliers*, valores ausentes ou a adaptação a um único *dataset*.

Ainda de acordo com Idri, Hosni e Abran (2016) para solucionar os problemas decorrentes de técnicas de AM individuais, foi proposto o uso de técnicas *ensemble*, que é a combinação de duas ou mais técnicas de AM individuais sob algum critério. A partir de então, surgiu outro problema para os pesquisadores, por exemplo, quais técnicas individuais combinar para criar um *ensemble*, ou ainda, quais critérios adotar na criação do *ensemble*. De acordo com Azzeh, Nassif e Minku (2015) um *ensemble* deve ser construído a partir

de técnicas de AM individuais que deem estimativas pobres para exemplos diferentes. Assim, uma técnica de AM que tenha estimativas ruins para um determinado *dataset* é compensada por outra técnica de AM com resultados bons para o mesmo *dataset*.

Pospieszny, Czarnacka-Chrobot e Kobylinski (2018) propuseram a criação de um *ensemble* utilizando três técnicas de AM, *Support Vector Machine* (SVM), *Multi-Layer Perceptron* (MLP) e *Generalized Linear Models* (GLM). Segundo os autores o SVM foi escolhido por gerar resultados com grande acurácia para problemas lineares e não lineares e ainda ser robusto quanto a dados ausentes. Já o MLP foi escolhido pela convergir ao identificar o mínimo local em vez do global. Por fim, o GLM foi escolhido por representar os relacionamentos entre os atributos. O *dataset* escolhido foi o ISBSG (*International Software Benchmarking Standards Group*), que é um *dataset* constituído por projetos de diversas empresas ao redor do mundo. Um ponto a ser observado é que foram utilizados softwares desenvolvidos com vários métodos, como Pontos por Função, Pontos por Caso de Uso, métodos ágeis entre outros. Os resultados foram avaliados utilizando as métricas MMRE, MAE, PRED(25) entre outras, apresentando um resultado preciso em relação a outras técnicas da literatura de acordo com os autores. Sendo uma das principais limitações a utilização de apenas um *dataset*.

Já Satapathy, Acharya e Rath (2016) decidiram analisar uma técnica *ensemble* popular na literatura, o *Random Forest* (RF), empregada a um contexto especifíco que foram softwares baseados em Pontos por Caso de Uso. Este método foi escolhido por ser amplamente utilizado na última década, simplicidade, rapidez e relativa precisão. Contudo, foi reportado a dificuldade de encontrar um *dataset* que tenha dado unicamente de projetos com esta metodologia. Os resultados foram avaliados utilizando as métricas MMRE, PRED(25). Os resultados foram comparados com outros da literatura apresentando resultados mais precisos que os demais quanto a estimativa de esforço no contexto escolhido.

Por fim Jodpimai, Sophatsathit e Lursinsap (2018) propuseram a criação de um *ensemble* para a estimativa de esforço utilizando algoritmos genéticos. Foram utilizados o *Ordinary Least Square Regression* (OLS), *Rregression Tree* (RT) e *K-Nearest Neighbour* (KNN), estes dois últimos são considerados o estado da arte para estimativa esforço (SONG; MINKU; YAO, 2018). Uma das maiores foi a proposição do algoritmo genético para seleção dos melhores algoritmos, que, foram selecionados de acordo com o menor erro para a métrica MMRE. Os resultados foram comparados com outros da literatura através das métricas MMRE, MdMRE e PRED(25), e mostram que o *ensemble* proposto superou os resultados da literatura de acordo com os autores.

3.2 Estimativas com AutoML

O AutoML surgiu com a quantidade crescente de dados armazenados por empresas e falta de mão obra especializada para tratar estes dados e extrair informações úteis (BALAJI; ALLEN, 2018). Nos últimos anos foram propostas algumas técnicas de AutoML como Auto-Weka, Auto-Sklearn, TPOT dentre outras. Estas técnicas têm ganhado notoriedade na literatura e possuem competições próprias (BALAJI; ALLEN, 2018) e estão sendo aplicadas a diversas áreas do conhecimento. Segundo Balaji e Allen (2018) as técnicas AutoML ainda não se mostraram mais eficiente que as produzidas manualmente por especialistas, no entanto, estas demandam muito menos esforço e em alguns casos têm apresentado resultados similares ao estado da arte (TANAKA; MONDEN; YÜCEL, 2019).

O uso de AutoML para problemas da engenharia de software ainda é vago, principalmente no que diz respeito a estimativa de esforço. Na literatura foi reportado o emprego de AutoML, em específico o Auto-Sklearn para a predição de defeitos no desenvolvimento de software (TANAKA; MONDEN; YÜCEL, 2019). Segundo o autor os resultados obtidos foram similares ao estado da arte do problema estudado.

3.3 Considerações Finais

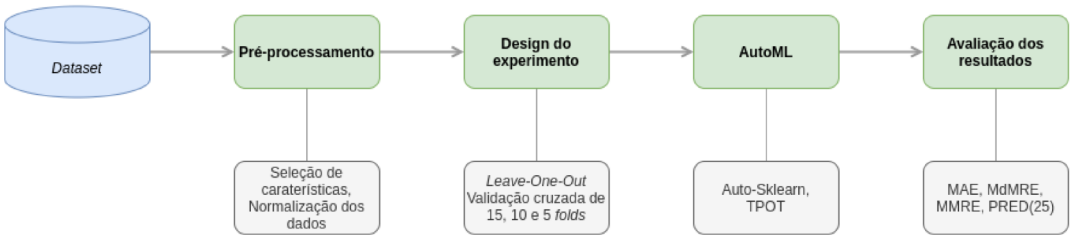
Como foi apresentado, a estimativa de esforço é um tema recorrente dentro da engenharia de software. Ao longo de mais de duas décadas foram propostas diversas técnicas de AM com objetivo de obter estimativas mais precisas, no entanto, este campo ainda requer mais estudos.

Os trabalhos apresentados neste capítulo abordaram a estimativa de esforço usando *ensemble* de técnicas de AM, os quais têm obtido melhores resultados neste campo. No entanto, embora tenham obtido algum progresso, ainda não existe uma técnica que se destaque das demais o que sugere que devem ser propostas novas técnicas *ensemble*, ou ainda, a aplicação de AutoML que trazem consigo uma gama de técnicas de AM e tem-se mostrado capaz de atingir resultados satisfatórios em outras áreas de pesquisa.

4 Metodologia

Este capítulo apresenta a metodologia proposta para estimativa de esforço utilizando AutoML. Esta metodologia expõe como as técnicas e conceitos introduzidos no Capítulo 2 são aplicados nesta pesquisa. A Figura 6 apresenta um resumo das etapas presentes na metodologia.

Figura 6 – Metodologia proposta



Fonte: Acervo do autor

Inicialmente é apresentado como foi realizado o processo de pré-processamento dos *datasets*. Posteriormente são descritos os designs de experimentos usados nesta pesquisa, bem como se deu a etapa de avaliação dos resultados.

4.1 Pré-processamento

A etapa de pré-processamento dos dados tem o objetivo de garantir que sejam usados apenas características que tenham relação com o esforço no desenvolvimento de software. Além disso, caso existam valores ausentes, estes também devem ser tratados para amenizar sua influência nos resultados. Por fim, foi realizada também a normalização dos dados.

4.1.1 Seleção de caraterísticas

A seleção das características é uma etapa importante, pois, deve garantir que as variáveis usadas nos experimentos tenham relação com o esforço do software. Nesta etapa cada um dos *datasets* foi analisado individualmente para verificar quais características deveriam ser usadas. Os *datasets* usados neste trabalho são constituídos de projetos desenvolvidos usando as metodologias COCOMO, Pontos de Função, Pontos de Caso de Uso e Web (MENDES et al., 2007). Então, para evitar a subjetividade, foram usadas as características que eram descritas em cada um destes modelos. *Datasets* de projetos feitos seguindo o COCOMO, foram usadas características relativas ao modelo COCOMO, assim,

como os demais *datasets*. Para o *dataset* Tukutuku, que possui características relativas ao contexto de projetos desenvolvido para Web, foram adotadas as características relatadas pelos autores (MENDES et al., 2007). Assim, evitou-se o uso de características que fossem derivadas do esforço real do projeto. Por exemplo, o *dataset* China (Pontos de Função), possui além do esforço real de cada um dos sistemas, o tempo de duração do projeto que é derivado do esforço real. Logo, a duração do projeto não poderia ser adota nos experimentos, uma vez que, este é um dado que não se tem ao realizar uma estimativa de um projeto real.

Continuando a análise dos dados, foram verificados a existência de valores ausentes nos *datasets*. Segundo Cartwright, Shepperd e Song (2004), Shah et al. (2019) as maneiras mais comuns de lidar com valores ausentes em *datesets* de engenharia de software são: (1) imputando o valor médio da mesma característica nos outros projetos; (2) um valor arbitrário; (3) a remoção do projeto que possui um ou mais valores ausentes; ou (4) desconsiderar a característica que esta ausente em um ou mais projetos. Ao introduzir um valor nos dados ausentes, por melhor que sejam os métodos usados, pode-se está enviesando os resultados (SHAH et al., 2019). A remoção de projetos que possuam dados ausentes foi desconsiderada devido aos *datsets* de engenharia de software já conterem um pequeno quantitativo de dados (TIERNO; NUNES, 2013), assim, esse método foi desconsiderado uma vez que deseja-se dar ao AutoML a maior quantidade de dados possíveis para a etapa de treinamento. Por fim, optou-se por remover as características que estejam ausentes em um ou mais projetos.

Ainda na análise dos dados foram analisados se os *datasets* possuíam valores categóricos, em outras palavras, valores não numéricos. Caso houvessem, era verificado a existência de um valor numérico correspondente na literatura. Por exemplo, o *dataset* Nasa possui algumas das características descritas como as categorias do modelo COCOMO (muito baixo, baixo, normal, alto e muito alto). Os valores categóricos foram então substituídos por seus correspondentes numéricos do modelo COCOMO (WAZLAWICK, 2013). Entretanto, outras categorias encontradas nos *datasets*, por exemplo, a linguagem de programação usada no desenvolvimento dos sistemas foram descartadas, assim, como todas as características que não estão relacionadas a estimativa de esforço, por exemplo, ano de desenvolvimento, número do projeto no *dataset*, data da estimativa, código do cliente, entre outros.

4.1.2 Normalização dos dados

De acordo com Song, Minku e Yao (2018) os valores de esforço dos *datasets* tendem a ser muito dispersos, isto é, existem alguns poucos projetos que têm o valor de esforço muito acima dos demais, ou ainda muito abaixo (MENZIES; GREENWALD; FRANK, 2006). Estes valores de esforços desbalanceados tendem a dificultar que a técnica AM consiga

estabelecer uma relação entre as características do projeto e o esforço (MARSLAND, 2015). Assim, foi necessário aplicar uma transformação para que os valores de esforço se tornassem menos dispersos. Segundo (SONG; MINKU; YAO, 2018), a aplicação da função logarítmica para *datasets* de estimativa de esforço fazem com que os dados sejam distribuídos de maneira normalizada e menos dispersos. Segundo os mesmos autores, a aplicação da função logarítmica não prejudica os resultados, e por vezes melhora o desempenho da estimativa de esforço.

Finalmente, os valores de esforços foram substituídos por seus valores de logaritmos naturais. Tomou-se o cuidado ainda de verificar a existência de projetos de valor de esforço 1 ou menor, uma vez que a função logarítmica para 1 é 0, e para valores menores que 0 é negativo. Como o esforço não pode assumir um valor negativo, os projetos que apresentassem o menor valor de esforço em cada *dataset* foram verificados e caso o esforço fosse menor ou igual a um, eram somados o valor da diferença entre o esforço do menor projeto e 2 em todos era somado a diferença entre valor de esforço deste projeto e 2 para todos os valores de esforços do *dataset*.

4.2 Design dos Experimentos

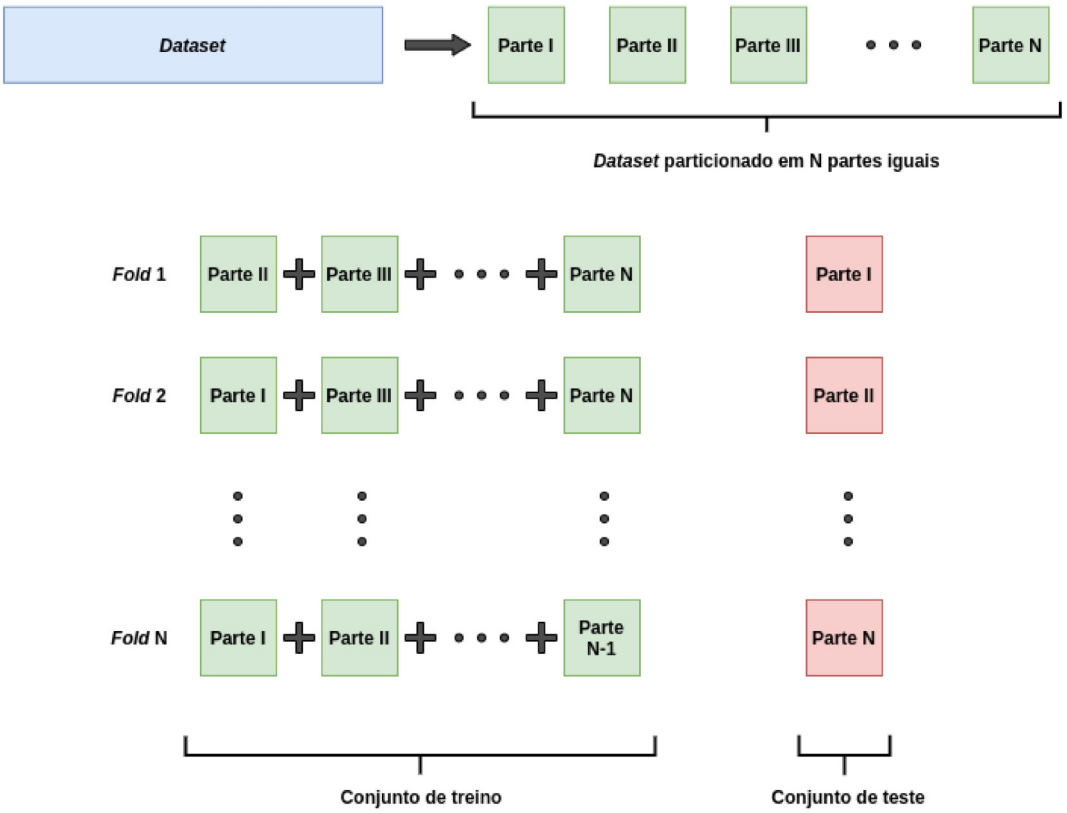
Os experimentos foram conduzidos após a etapa de pré-processamento dos *datasets*. Os experimentos foram avaliados usando validação cruzada. A quantidade de *folds* foi definida como 15, 10, 5 e (quantidade de indivíduos -1). Este último caso é um comportamento especial da validação cruzada chamada *Leave-one-out*, caracterizada por ser usada quando se trata de bases pequenas.

A validação cruzada com *k-folds* é apresentada na Figura 7. Nos experimentos com validação cruzada, o *dataset* é dividido em k *folds* mutuamente exclusivos de tamanho aproximadamente igual (igual quando o número de dados for múltiplo de k). Após a criação dos subconjuntos são utilizados todos os subconjuntos para treinamento menos um, que é utilizado na etapa de teste. O tamanho do conjunto de teste, é inversamente proporcional ao valor de k, assim, quanto menor o valor de k, maior o tamanho do conjunto de teste, portanto, menos dados são utilizados na etapa de treinamento.

Nos experimentos que foram realizados usando validação cruzada os dados foram distribuídos aleatoriamente entre os subconjuntos. A aleatoriedade tem o objetivo de evitar a criação de subconjuntos que tenham os mesmo dados. Assim, pretende-se evitar que os resultados sejam tendenciosos.

Ao utilizar a validação cruzada *Leave-One-Out* é dado ao algoritmo de AM, o máximo de conhecimento possível acerca do conjunto de dados, enquanto na validação cruzada com *k-folds* de 5, é utilizado 4/5 do *dataset* para o treinamento o que representa a menor quantidade entre os experimentos realizados. Considerando que o tamanho dos

Figura 7 – Validação cruzada com *k-folds*



Fonte: Acervo do autor

datasets de estimativa de esforço são em sua maioria historicamente pequenos (TIERNO; NUNES, 2013), diminuir o tamanho da base de treino pode levar a estimativas com maior grau de imprecisão.

4.3 AutoML

Para os experimentos realizados neste trabalho, foram usadas as distribuições oficiais do Auto-Sklearn e TPOT. Em relação aos parâmetros que podem ser configurados para ambos os AutoMLs, foram usados os valores padrões de ambos. O Auto-Sklearn pode ser configurado em relação à quantidade de tempo que será executado, quanto mais tempo é dado, melhor tendem ser as estimativas geradas. Já o TPOT por usar algoritmos genéticos, podem ser definidos o número de gerações, assim como, o tamanho da população, e quanto maior o número de gerações melhor tendem a ser as estimativas, contudo, o tempo de execução será maior. Por não haver nenhuma configuração já estabelecida dessas técnicas de AutoML para a estimativa de esforço, optou-se por usar os valores padrões.

O tempo de execução do Auto-Sklearn e TPOT é aproximadamente o mesmo para cada *dataset*, como não estava entre os objetivos do trabalho fazer uma análise em relação

ao tempo de execução, estes dados não foram mensurados. Entretanto, considerando que o tempo de execução padrão do Auto-Sklearn é de 120 segundos para uma instância e considerado que esse tempo será aplicado para cada projeto de um *dataset*. O tempo de execução do Auto-Sklearn para o maior *dataset*, China, que possui 499 projetos, para o design *Leave-One-Out* foi cerca de 17 horas. Já para o menor *dataset*, Kemerer, o tempo de execução é em média 30 minutos.

4.4 Avaliação dos resultados

A avaliação dos resultados foi realizada aplicando as métricas apresentadas na subseção 2.1.4. As métricas foram escolhidas devido à representatividade na literatura (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016) permitindo assim uma posterior comparação com outros trabalhos.

É valido ainda ressaltar que os valores para as métricas MAE, MMRE e MdMRE quanto menores forem mais próximo ao esforço real, portanto, melhores são as estimativas. Já para a métrica PRED(25) quanto maior o valor melhor. Segundo Wen et al. (2012), Idri, Hosni e Abran (2016) estimativas consideradas aceitáveis o valor da métrica MMRE deve ser menor ou igual a 25% e para PRED(25) o valor deve ser acima dos 75%. Os valores tidos como aceitáveis pela literatura para MMRE e PRED(25) foram considerados na avaliação dos resultados.

A apresentação dos resultados foi realizada de acordo com os métodos de coleta dos *datasets*, apenas para fins de organização.

4.5 Considerações finais

Este capítulo descreveu a metodologia proposta para estimativa de esforço usando AutoML. Foram apresentados todos os passos presentes na metodologia, e como foram executados, considerando as características de cada *dataset* e projeto com objetivo de extrair o máximo de dados para o treinamento dos AutoMLs usados para as estimativas. A seguir, tem-se o detalhamento da execução dos experimentos realizados com a metodologia proposta usando as duas técnicas de AutoML escolhidas e uma análise do impacto da metodologia para a estimativa de esforço.

5 Experimentos e discussão

Este capítulo discorre sobre a execução dos experimentos, desde o pré-processamento até a avaliação dos resultados. Além disso, é feito uma comparação com outros trabalhos da literatura os quais podem ser considerados o estado da arte da estimativa de esforço utilizando AM. Por fim, são introduzidas as principais ameaças a validade desta pequisa, assim como, as medidas adotadas para minimizá-las.

5.1 *Datasets*

Um dos principais objetivos deste trabalho é criar uma metodologia robusta, capaz de realizar estimativas coerentes para projetos desenvolvidos em diversos métodos. Foram coletados assim *datasets* que pudessem representar tal diversidade, os *datasets* coletados possuem projetos mensurados usando Pontos por Caso de Uso, Pontos por Função, entre outros.

A maioria dos *datasets* são oriundos do repositório PROMISE (SHIRABAD; MENZIES, 2005) que é amplamente usado em trabalhos de estimativa de esforço (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016). Já outros *datasets* foram cedidos por seus autores. Foram usados 10 *datasets* durante este trabalho e são apresentados brevemente na Tabela 4.

Os *datasets* usados nesta pesquisa são de projetos mensurados usando os métodos COCOMO(2), Pontos de Função (6) e Pontos de Caso de Uso (1), com exceção do *dataset* Tukutuku que possui dados de projetos mensurados seguindo algumas métricas próprias de projetos Web (MENDES et al., 2007).

5.2 Descrição do Experimento

Os experimentos foram executados de acordo seguindo os passos descritos no Capítulo 4. Foi feito o pré-processamento dos dados, em seguida foram configurados e executados os experimentos conforme os designes de experimentos expostos no capítulo anterior.

5.2.1 Pré-processamento

A atividade de pré-processamento ocorreu em duas etapas. Primeiramente foi realizado um estudo sobre os *datasets* usados, onde foi verificado individualmente quais das características anotadas deveriam ser usadas no experimento. Foi considerado, portanto, se a característica tinha alguma relação com o valor de esforço. Caso não houvesse relação,

Tabela 4 – Características gerais do esforço real para os projetos dos *datasets*.

Dataset	**Nº proj.**	**Carac.**	**Dados de esforço**					
			Unidade	**Min**	**Max**	**Média**	**Mediana**	**Dsv. pad.**
Albrecht	24	7	homem/mês	0,50	105,20	21,88	11,45	27,82
Cocomo81	63	16	homem/mês	5,90	11400,00	683,32	98,00	1807,07
China	499	18	homem/hora	26,00	5462,00	3921,05	1829,00	6474,36
Desharnais	81	12	homem/hora	546,00	23940,00	5046,31	3647,00	4391,41
Kemerer	15	6	homem/mês	23,20	1107,31	219,25	130,30	254,14
2 Maxwell	62	27	homem/hora	583,00	63694,00	8223,21	5189,50	10414,88
Miyazaki94	48	8	homem/hora	5,60	1586,00	87,47	38,10	226,36
Nasa	92	23	homem/mês	8,40	8211,00	624,41	252,00	1129,80
Tukutuku	195	22	homem/hora	1,10	5000,00	468,11	88,00	936,10
UCP	70	16	homem/hora	5775,00	7970,00	6571,27	6412,00	663,16

Fonte: Elaborada pelo autor

esta era desconsiderada. Características desconsideradas no *dataset* Kemerer, por exemplo, foram o identificador do projeto e a linguagem de desenvolvimento.

Após haverem sido selecionadas as características que deveriam ser usadas, foi analisado se para alguma destas havia algum projeto em que o valor estava ausente. Não foram encontrados dados ausentes em nenhum dos *datasets* usados, assim, não foi necessário remover nenhuma das características conforme detalhado na seção 4.1.

O segundo e último passo, foi feito de forma automatizada usando uma biblioteca Numpy[1]. A biblioteca foi usada para substituir o valor do esforço por seu correspondente na base de logaritmo natural. Após, este passo os dados estavam prontos para serem usados nos experimentos.

5.2.2 Ambiente de execução

Os experimentos foram realizados numa máquina com sistema operacional Linux, uma vez que, o Auto-Sklearn é passível de instalação unicamente neste sistema operacional. A Tabela 5 apresenta as configurações do ambiente usado para execução dos experimentos.

Tabela 5 – Configuração do ambiente de execução dos experimentos.

Hardware	
CPU	Intel(R) Core(TM) i3-9100F @ 3.60GHz
Memória	16,0 GB
Software	
Sistema Operacional	Ubuntu 18.04.4
Ferramentas	Visual Studio Code
Programação	
Linguagem	Python 3.7.4
Bibliotecas	Auto-Sklearn 0.9.0, TPOT

Fonte: Elaborada pelo autor

5.3 Resultados e Discussões

Os resultados dos experimentos são apresentados de acordo com os métodos de estimativa dos *datasets*. São apresentados os resultados para os *datasets* que contém métricas do método COCOMO, em seguida os *datasets* de Pontos de Função e por fim, optou-se por mostrar em conjunto os *datasets* que foram mensurados com Ponto de Função e métricas Web por terem *dataset* cada um.

[1] https://numpy.org/

5.3.1 *Datasets* COCOMO

Os *datasets* que possuem projetos desenvolvidos seguindo as métricas do modelo COCOMO são Cocomo81 e Nasa. Os resultados dos experimentos para estes *datasets* são apresentados nas Tabelas 6, 7, 8 e 9. Estão destacados em negritos os melhores resultados para cada uma das métricas considerando os AutoMLs.

Tabela 6 – Resultados para *datasets* COCOMO: Resultados da metodologia para o *Leave-One-Out*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
Cocomo81	Auto-Sklearn	**0,53**	**8,83**	**13,03**	**85,71**
	TPOT	0,63	12,36	14,68	84,13
Nasa	Auto-Sklearn	**0,51**	**6,89**	**10,17**	**90,32**
	TPOT	0,60	7,48	12,02	87,10

Fonte: Elaborada pelo autor

Tabela 7 – Resultados para *datasets* COCOMO: Média e desvio padrão para os experimentos usando validação cruzada de 15 *folds*

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Cocomo81	Auto-Sklearn	1,12±0,08	18,45±0,85	28,57±2,10	65,40±2,84
	TPOT	**0,59±0,00**	**11,17±0,00**	**14,66±0,00**	**87,30±0,00**
Nasa	Auto-Sklearn	0,97±0,00	14,86±0,00	19,17±0,00	74,19±0,00
	TPOT	**0,54±0,02**	**7,65±0,36**	**10,87±0,21**	**91,40±0,00**

Fonte: Elaborada pelo autor

Tabela 8 – Resultados para *datasets* COCOMO: Média e desvio padrão para os experimentos usando validação cruzada de 10 *folds*.

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Cocomo81	Auto-Sklearn	1,14±0,01	20,44±0,17	28,04±0,18	64,44±0,87
	TPOT	**0,68±0,00**	**13,12±0,00**	**15,39±0,00**	**82,54±0,00**
Nasa	Auto-Sklearn	1,02±0,04	15,36±0,48	19,91±0,73	67,74±2,41
	TPOT	**0,61±0,01**	**10,19±0,00**	**11,80±0,13**	**91,40±0,00**

Fonte: Elaborada pelo autor

Na Tabela 6 considerando as métricas de MMRE e PRED(25) os resultados estiveram na margem de aceitação usando os dois AutoMLs. Com destaque para o Auto-Sklearn que teve um desempenho ligeiramente melhor que o TPOT. O valor de MMRE manteve-se abaixo dos 14% indicando que a média dos erros relativos foi baixa. Já o menor valor de PRED(25) foi 84,13%.

Em relação aos experimentos com validade cruzada, maiores são os erros calculados para todas em todas as métricas. O desvio padrão foi mais constante nos resultados do

Tabela 9 – Resultados para *datasets* COCOMO: Média e desvio padrão para os experimentos usando validação cruzada de 5 *folds*

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Cocomo81	Auto-Sklearn	1,23±0,29	20,99±4,37	30,80±8,12	60,63±11,35
	TPOT	**0,82±0,00**	**14,66±0,00**	**20,00±0,00**	**68,25±0,00**
Nasa	Auto-Sklearn	0,96±0,17	15,90±4,19	19,04±3,50	72,04±10,47
	TPOT	**0,72±0,00**	**9,53±0,00**	**13,32±0,00**	**89,25±0,00**

Fonte: Elaborada pelo autor

TPOT, enquanto o Auto-Sklearn teve uma variação um pouco maior, chegando a variar 11,35% para métrica de PRED(25) na Tabela 9, o que indica que em algumas execuções o erro pode ter sido abaixo dos 50%.

Pode ser observado ainda que quanto menos foram usados na etapa de treinamento, maiores foram os erros apresentados nos dois *datasets*. Entretanto, considerando uma situação de uso na indústria é usado a maior conjunto de dados possível para o treinamento, semelhante ao *Leave-One-Out*, pode se dizer então que ambos os AutoML conseguiram estimativas dentro das margens consideradas com aceitáveis.

5.3.2 *Datasets* de Pontos de Função

A maioria dos *datasets* usados são compostos por projetos que foram desenvolvidos usando a métrica de Pontos de Função. Os resultados dos experimentos realizados para estes *datasets* são apresentados nas Tabelas 10, 11, 12 e 13. Foram destacados os melhores resultados para cada uma das métricas.

Tabela 10 – Resultados para *datasets* de Pontos de Função: Resultados da metodologia para o *Leave-One-Out*

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Albrecht	Auto-Sklearn	**0,42**	11,60	21,22	79,17
	TPOT	**0,42**	**11,57**	**22,20**	**83,33**
China	Auto-Sklearn	0,16	1,35	2,26	**99,80**
	TPOT	**0,13**	**1,08**	**1,79**	**99,80**
Desharnais	Auto-Sklearn	0,39	3,94	4,80	**100,00**
	TPOT	**0,38**	**3,81**	**4,62**	**100,00**
Kemerer	Auto-Sklearn	**0,47**	**8,75**	**9,60**	**100,00**
	TPOT	0,55	10,14	11,70	93,33
Maxwell	Auto-Sklearn	**0,65**	**6,26**	**7,91**	**96,77**
	TPOT	0,71	7,57	8,58	**96,77**
Miyazaki94	Auto-Sklearn	**0,44**	**8,61**	**12,98**	**89,58**
	TPOT	0,50	9,87	14,79	87,50

Fonte: Elaborada pelo autor

Tabela 11 – Resultados para *datasets* de Pontos de Função: Média e desvio padrão para os experimentos usando validação cruzada de 15 *folds*

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Albrecht	Auto-Sklearn	0,49±0,00	13,57±0,00	24,38±0,01	70,83±0,00
	TPOT	**0,41±0,00**	**11,43±0,00**	**22,44±0,00**	**83,33±0,00**
China	Auto-Sklearn	0,45±0,00	4,77±0,00	6,12±0,00	98,40±0,00
	TPOT	**0,14±0,00**	**1,24±0,00**	**1,98±0,00**	**99,80±0,00**
Desharnais	Auto-Sklearn	**0,45±0,01**	**4,55±0,12**	**5,65±0,22**	**100±0,00**
	TPOT	0,51±0,00	5,55±0,01	6,39±0,00	**100±0,00**
Kemerer	Auto-Sklearn	0,61±0,02	**7,63±0,00**	12,00±0,36	91,99±2,97
	TPOT	**0,49±0,00**	7,94±0,00	**9,63±0,00**	**100±0,00**
Maxwell	Auto-Sklearn	0,87±0,05	**9,06±0,68**	10,46±0,57	95,16±0,00
	TPOT	**0,84±0,02**	9,15±0,44	**10,07±0,29**	**97,09±0,72**
Miyazaki94	Auto-Sklearn	**0,55±0,02**	14,27±1,41	**17,05±0,98**	**80,83±2,71**
	TPOT	0,63±0,00	**11,15±0,00**	17,36±0,00	70,83±0,00

Fonte: Elaborada pelo autor

Tabela 12 – Resultados para *datasets* de Pontos de Função: Média e desvio padrão para os experimentos usando validação cruzada de 10 *folds*

Dataset	**AutoML**	**MAE**	**MdMRE%**	**MMRE%**	**PRED(25)%**
Albrecht	Auto-Sklearn	0,41±0,00	**11,35±0,00**	**21,05±0,02**	**83,33±0,00**
	TPOT	**0,39±0,00**	11,73±0,00	21,10±0,00	79,17±0,00
China	Auto-Sklearn	0,45±0,00	4,78±0,00	6,18±0,00	**98,80±0,00**
	TPOT	**0,14±0,00**	**1,38±0,00**	**1,89±0,01**	**99,80±0,00**
Desharnais	Auto-Sklearn	**0,41±0,01**	**4,10±0,05**	5,18±0,14	**100,00±0,00**
	TPOT	0,42±0,00	4,77±0,00	**5,29±0,00**	**100,00±0,00**
Kemerer	Auto-Sklearn	0,71±0,00	13,04±0,37	14,25±0,07	86,67±0,00
	TPOT	**0,59±0,00**	**10,09±0,00**	**12,01±0,00**	**93,33±0,00**
Maxwell	Auto-Sklearn	0,82±0,00	8,22±0,00	9,79±0,00	**96,77±0,00**
	TPOT	**0,73±0,01**	**8,01±0,00**	**8,86±0,19**	**96,77±0,00**
Miyazaki94	Auto-Sklearn	0,61±0,01	14,04±1,11	17,67±0,39	87,08±0,93
	TPOT	**0,45±0,00**	**8,44±0,00**	**12,72±0,00**	**89,58±0,00**

Fonte: Elaborada pelo autor

Tabela 13 – Resultados para *datasets* de Pontos de Função: Média e desvio padrão para os experimentos usando validação cruzada de 5 *folds*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
Albrecht	Auto-Sklearn	0,54±0,01	14,66±0,59	23,21±0,37	70,83±0,00
	TPOT	**0,39±0,00**	**11,35±0,00**	**21,65±0,00**	**83,33±0,00**
China	Auto-Sklearn	0,49±0,00	5,17±0,00	6,65±0,00	98,40±0,00
	TPOT	**0,13±0,00**	**1,08±0,00**	**1,72±0,00**	**99,80±0,00**
Desharnais	Auto-Sklearn	0,43±0,01	**3,81±0,04**	5,45±0,12	**100,00±0,00**
	TPOT	**0,41±0,00**	4,04±0,00	**5,07±0,00**	**100,00±0,00**
Kemerer	Auto-Sklearn	0,74±0,00	12,85±0,00	14,47±0,02	**93,33±0,00**
	TPOT	**0,54±0,00**	**10,14±0,00**	**11,63±0,14**	**93,33±0,00**
Maxwell	Auto-Sklearn	0,78±0,00	**6,99±0,00**	9,37±0,02	93,55±0,00
	TPOT	**0,75±0,01**	7,61±0,17	**9,19±0,15**	**95,16±0,00**
Miyazaki94	Auto-Sklearn	0,56±0,01	10,40±0,77	16,20±0,24	**85,42±0,00**
	TPOT	**0,49±0,00**	**9,00±0,00**	**13,38±0,00**	**85,42±0,00**

Fonte: Elaborada pelo autor

Do ponto de vista das métricas MMRE e PRED(25) os resultados apresentados nas Tabelas 10, 11, 12 e 13 são considerados bons. No *dataset* Albrecht encontram-se os maiores erros para os dois AutoML. É importante salientar também que os *datasets* que possuem a maior quantidade de dados e a menor, China e Kemerer respectivamente, mostram que a quantidade de dados não apresenta uma relação direta com os resultados.

Os resultados da Tabela 11 mostram uma perda de desempenho em relação aos experimentos de *Leave-One-Out*. Em dois casos a média para a métrica PRED(25) esteve abaixo dos 75%. A métrica MdMRE manteve-se abaixo dos 15% indicando que o erro médio relativo foi baixo nesses experimentos. O desvio padrão foi nulo para a maioria dos experimentos, o que se significa que mesmo criando subgrupos de forma aleatória, os erros estão se mantendo iguais.

5.3.3 *Datasets* de Pontos de Caso de Uso e Métricas Web

Os resultados para os *datasets* UCP e Tukutuku serão mostrados em conjunto, uma vez que existem um representante para cada uma de suas métricas. É importante frisar que mesmo *datasets* que contenham projetos com as mesmas métricas apresentam resultados diferentes. Isto porque os projetos coletados são diferentes.

Os resultados apresentados para o *dataset* UCP em foram bons para as métricas MMRE e PRED(25). Onde a média de PRED(25) foi de 100% e média do desvio padrão 0 nos experimentos que envolveram o Auto-Sklearn, o que significa que em todas as estimativas realizadas o erro cometido relativo esteve abaixo dos 25%. O Auto-Sklearn ainda conseguiu uma média abaixo de 1% para o MMRE, essas métricas indicam que os resultados estiveram muito próximo ao real valor do esforço.

Tabela 14 – Resultados para os *datasets* UCP e Tukutuku: Resultados da metodologia para o *Leave-One-Out*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
UCP	Auto-Sklearn	**0,03**	**0,19**	**0,31**	**100,00**
	TPOT	0,03	0,29	0,35	**100,00**
Tukutuku	Auto-Sklearn	**0,68**	**10,32**	30,16	**76,41**
	TPOT	0,69	11,49	**26,75**	75,90

Fonte: Elaborada pelo autor

Tabela 15 – Resultados para os *datasets* UCP e Tukutuku: Média e desvio padrão para os experimentos usando validação cruzada de 15 *folds*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
UCP	Auto-Sklearn	0,05±0,00	0,43±0,00	0,58±0,00	**100,00±0,00**
	TPOT	**0,04±0,00**	**0,33±0,00**	**0,44±0,00**	**100,00±0,00**
Tukutuku	Auto-Sklearn	0,96±0,00	18,39±0,18	44,29±2,87	68,92±1,83
	TPOT	**0,80±0,00**	**14,01±0,00**	**32,68±0,00**	**70,77±0,00**

Fonte: Elaborada pelo autor

Tabela 16 – Resultados para os *datasets* UCP e Tukutuku: Média e desvio padrão para os experimentos usando validação cruzada de 10 *folds*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
UCP	Auto-Sklearn	0,07±0,00	0,68±0,00	0,76±0,00	**100,00±0,00**
	TPOT	**0,04±0,00**	**0,40±0,00**	**0,48±0,00**	**100,00±0,00**
Tukutuku	Auto-Sklearn	1,09±0,00	19,92±0,00	55,95±0,00	62,56±0,00
	TPOT	**0,87±0,00**	**15,83±0,00**	**39,49±0,00**	**72,82±0,00**

Fonte: Elaborada pelo autor

Tabela 17 – Resultados para os *datasets* UCP e Tukutuku: Média e desvio padrão para os experimentos usando validação cruzada de 5 *folds*

Dataset	AutoML	MAE	MdMRE%	MMRE%	PRED(25)%
UCP	Auto-Sklearn	0,06±0,01	**0,44±0,02**	**0,72±0,08**	**100±0,00**
	TPOT	**0,05±0,00**	9,54±20,31	12,37±26,34	82,81±38,42
Tukutuku	Auto-Sklearn	1,05±0,03	20,01±1,67	53,79±0,10	63,07±2,29
	TPOT	**0,94±0,00**	**17,13±0,00**	**51,98±0,00**	**68,72±0,00**

Fonte: Elaborada pelo autor

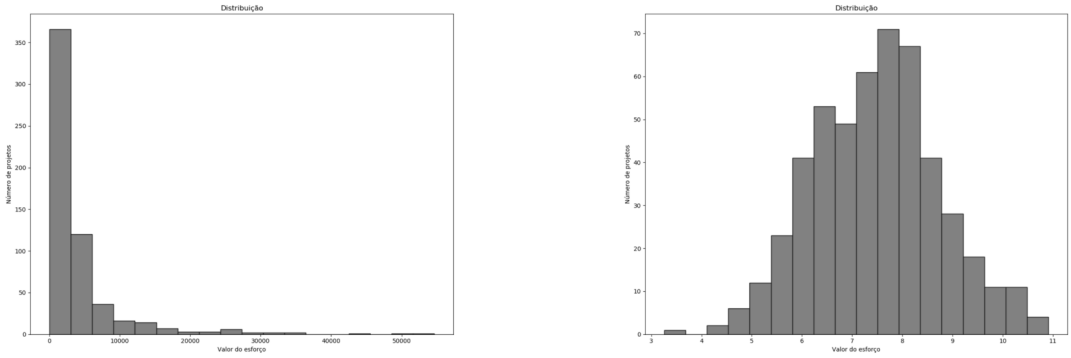
Em contrapartida, foram os experimentos envolvendo o *dataset* Tukutuku resultaram nos maiores erros em relação às métricas MdMRE, MMRE e PRED(25). Neste *dataset* nenhum dos AutoML testados conseguiram um MMRE menor que 25%. Todavia, a tabela Tabela 14 mostra que o valor de PRED(25) ficou acima dos 75%.

5.3.4 Discussão

A primeira etapa da metodologia proposta foi o pré-processamento dos dados. Ao aplicar a transformação de logaritmo natural deseja-se fazer a normalização dos *datasets*. As Figuras 8 e 9 apresentam os resultados da normalização para os *datasets* China e Desharnais respectivamente.

Figura 8 – Distribuição amostral dos valores de esforço para o *dataset* China antes e depois da normalização.

(a) Valor esforço real (b) Valor de esforço na base logarítmica



Fonte: Elaborada pelo autor

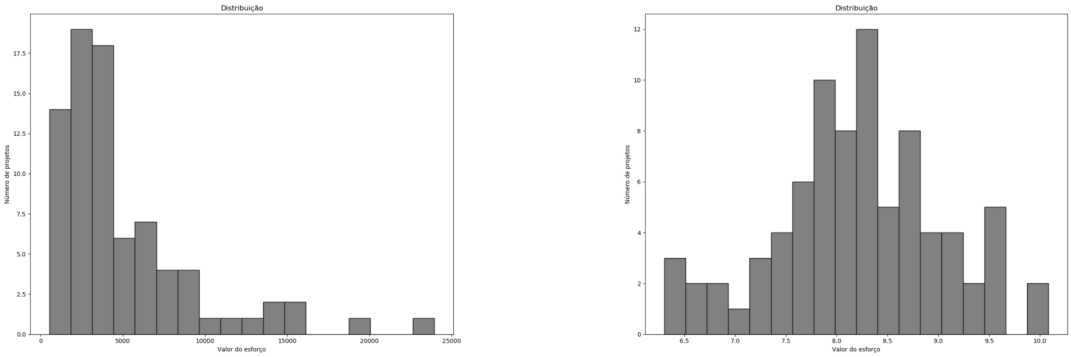
Os resultados do pré-processamento apresentados nas Figuras 8 e 9 mostram que a aplicação da função logarítmica na base natural conseguiu normalizar a distribuição dos *datasets*, que foi o principal objetivo ao realizar essa transformação.

Os resultados dos experimentos mostram que a metodologia, na maioria dos casos, conseguiu realizar estimativas apropriadas considerando as métricas MMRE e PRED(25). No que diz respeito as técnicas de AutoML, Auto-Sklearn e TPOT, os resultados mostram que o TPOT teve uma performance ligeiramente melhor em 27 dos 40 experimentos realizados. Tais resultados podem encontrar respostas na literatura que indica que o TPOT realiza estimativas melhores para problemas de regressão (BALAJI; ALLEN, 2018).

Figura 9 – Distribuição amostral dos valores de esforço para o *dataset* Desharnais antes e depois da normalização.

(a) Valor esforço real

(b) Valor de esforço na base logarítmica



Fonte: Elaborada pelo autor

Entretanto, os experimentos com Auto-Sklearn teve resultados bons, considerando que em apenas 11 ocasiões os resultados estiveram abaixo de 75% para a métrica PRED(25).

Pode ainda ser observado na Tabela 10 que as estimativas diferem mesmo para *datasets* que considerem a coleta dos mesmas características de projetos de software. Essa diferença entre os *datasets* é um dos maiores desafios no uso de técnicas de AM, pois, essas se comportam de forma diferente para cada *dataset* (IDRI; HOSNI; ABRAN, 2016). Apesar das diferenças nos resultados, a metodologia mostrou capaz de adaptar-se aos diferentes *datasets* gerando erros baixos para as métricas usadas.

Em relação ao design dos experimentos, os resultados mostram que metodologia proposta obteve melhores resultados para o *Leave-One-Out* nos dois AutoMLs. Isso pode ser comprovado observando que a métrica PRED(25) ficou acima dos 75% em todos os casos, e a MMRE ficou abaixo dos 25% para a maioria dos *dataset*, a exceção ocorreu no *dataset* Tukutuku. Ao considerar que neste modelo o treinamento tem a maior quantidade de dados possíveis, em relação aos demais, as técnicas de AutoML conseguem se adaptar melhor aos *datasets* neste modelo.

Nos experimentos de validação cruzada, o desvio padrão foi baixo, na grande maioria dos experimentos o desvio foi de zero para todas as métricas, o que indica que as técnicas usadas fizeram estimativas com resultados consistentes, uma vez que tiveram a mesma taxa de erro para conjuntos de treinamento e teste diferentes.

Os resultados também levam a concluir que a metodologia apresentada com ambos AutoMLs, consegue resultados aceitáveis para as métricas MMRE e PRED(25) mesmo para *datasets* com poucos dados, caso do Kemerer, o que é importante ao considerar que *datasets* de estimativa de esforço são historicamente pequenos (TIERNO; NUNES, 2013).

Os resultados dos experimentos mostraram que o número de projetos nos *datasets* não está relacionado diretamente a qualidade das estimativas geradas para a metodologia proposta. O segundo maior *dataset*, Tukutuku, apresentou os maiores erros de estimativa. A qualidade da estimativa pode estar mais relacionada a quão bem as características coletadas representam o esforço real que o número de projetos. Isso porque, por melhor que os valores para as características representem a realidade, algumas características são completamente subjetivas, por exemplo, a experiência da equipe de desenvolvimento.

Outro fator que pode ter influenciado os resultados no *dataset* Tukutuku é fato deste ser composto por projetos de múltiplas empresas. Embora este não seja um fato determinante na performance das técnicas de AM, Kitchenham, Mendes e Travassos (2007) conduziram uma revisão da literatura, e tiveram indícios que técnicas de AM têm melhor performance em *datasets* de única empresa que *datasets* compostos por dados de projetos oriundos de mais de uma empresa.

Assim, os resultados estão mais ligados aos dados coletados em si. É importante considerar este fato porque apenas as coletas dos dados já podem enviesar os resultados das estimativas geradas por técnicas de AM. Considerando que os dados possuem a subjetividade de quem coleta, as técnicas de AM, irão se comportar de maneira diferente de um *dataset* para outro. Apesar disso, as técnicas de AutoML usadas mostraram-se robustas gerando estimativas com erros baixos para os *datasets* estudados.

Ademais, considerando que a literatura define que estimativas com MMRE$\leq$25% e PRED(25)$\geq$75% (IDRI; HOSNI; ABRAN, 2016). A metodologia proposta apresenta indícios que pode generalizar boas estimativas para diferentes *datasets*. Essa generalização pode permitir que engenheiros de software usem a metodologia aplicada *datasets* locais, uma vez que os AutoML podem se auto ajustarem para o conjunto de dados usado. O uso de *datasets* de projetos internos de uma empresa usualmente geram estimativas mais precisas que *datasets* compostos por dados de mais de uma empresa (KITCHENHAM; MENDES; TRAVASSOS, 2007).

Já em relação aplicação da metodologia proposta na indústria existem algumas limitações. Primeiramente é necessário o engenheiro de software possua um conjunto de dados previamente coletado para que a técnica de AutoML possa aprender a partir destes e então realizar estimativas. Existe ainda a possibilidade de usar os modelos gerados com *datasets* genéricos, mas, estes podem não representar o melhor possível a realidade da empresa.

Outra limitação para aplicação na indústria é que o engenheiro de software caso resolva coletar dados dos projetos, deve ter um conhecimento prévio sobre estimativa de esforço para que saiba quais dados devem ser coletados, isto é, quais variáveis que influenciam o esforço. Assim, ainda existem algumas adversidades que devem ser superadas para criação de um produto final de estimativa de esforço.

Outrossim, deve ser considerado o impacto que os resultados podem representar para a qualidade dos softwares. Os resultados apresentam indícios que a metodologia pode vir a ser usada no auxílio de estimativas mais precisas, evitando desperdícios ou ainda escassez de recursos no desenvolvimento de softwares. Levando a construção de softwares com orçamento e cronograma de execução mais adequados.

5.3.5 Comparação com outros trabalhos de estimativa de esforço

A Tabela 18 apresenta uma comparação dos resultados da metodologia proposta com outros trabalhos encontrados na literatura. Para realização de uma comparação mais justa, foi escolhido os melhores resultados de um dos AutoML usados. Dado que as estimativas do TPOT para as métricas de MMRE e PRED(25) foram melhores que as do Auto-Sklearn em 23 dos 40 experimentos, foram escolhidos os resultados do TPOT para as comparações. As métricas usadas foram MdMRE, MMRE e PRED(25), a métrica MAE foi desconsiderada, pois, muitos dos trabalhos não apresentaram essa métrica. Não foram feitas comparações com o *dataset* UCP, uma vez que, os trabalhos que usaram este *dataset* (SILHAVY; SILHAVY; PROKOPOVA, 2015; AZZEH; NASSIF, 2018) não usaram as métricas de MMRE e PRED(25).

Apesar de alguns dos trabalhos usados na comparação terem sido publicados a mais de 5 anos, seus resultados continuam entre o estado da arte no que diz respeito a estimativa de esforço (IDRI; HOSNI; ABRAN, 2016; SONG; MINKU; YAO, 2018). Em relação à métrica MdMRE, a maioria dos trabalhos não usou essa métrica, mas, considerando unicamente os que publicaram, a metodologia proposta saiu-se melhor em 3 *datasets*.

Com relação à métrica MMRE a metodologia proposta destacou-se nos *datasets* Kemerer e Tukutuku, sendo que neste último o valor esteve acima dos 25%. Todavia, nos demais *datasets* os resultados se mantiveram constantemente abaixo dos 25% indicando que as estimativas geradas são aproximas ao esforço real.

Levando-se em consideração a métrica PRED(25), a metodologia proposta conseguiu resultados melhores que os outros trabalhos em 6 dos 9 *datasets*. Isso levanta indícios que a metodologia proposta usando AutoML consegue generalizar os resultados para diversas bases. Sendo os resultados mais generalistas que os dos métodos *ensemble*.

É importante salientar que a metodologia proposta mesmo não alcançando os

Tabela 18 – Comparação por *dataset* dos resultados da metodologia com outros trabalhos

Dataset	Trabalho	MdMRE%	MMRE%	PRED(25)%
Albrecht	(KUMAR et al., 2008)	**9,70**	**7,18**	**87,50**
	(ANGELIS; STAMELOS, 2000)	-	31,00	72,00
	(CHIU; HUANG, 2007)	19,00	33,00	70,00
	metodologia proposta	11,57	22,20	83,33
China	(KHAZAIEPOOR; BARDSIRI; KEYNIA, 2019)	0,11	0,23	75,33
	(KHAZAIEPOOR; BARDSIRI; KEYNIA, 2019)	0,07	0,19	80,00
	(KHAZAIEPOOR; BARDSIRI; KEYNIA, 2019)	**0,05**	**0,11**	84,00
	metodologia proposta	1,08	1,79	**99,80**
Cocomo81	(AMARAL et al., 2019)	-	14,13	79,36
	(NAGPAL; UDDIN; KAUR, 2012)	-	21,04	76,19
	(ARAUJO; OLIVEIRA; SOARES, 2010)	-	**12,81**	90,90
	metodologia proposta	**12,36**	14,68	**84,13**
Desharnais	(AMARAL et al., 2019)	-	**4,20**	**100,00**
	(NASSIF et al., 2013)	14,00	25,00	72,00
	(NAGPAL; UDDIN; KAUR, 2012)	-	9,81	90,00
	metodologia proposta	**3,81**	4,62	**100,00**
Kemerer	(AMARAL et al., 2019)	-	10,54	93,33
	(LOPES, 2019)	-	19,00	83,00
	(OLIVEIRA et al., 2010)	-	33,49	64,00
	metodologia proposta	-	**7,91**	**96,77**
Maxwell	(AMARAL et al., 2019)	-	**7,88**	**96,77**
	(ELISH; HELMY; HUSSAIN, 2013)	-	59,49	25,00
	(LI; XIE; GOH, 2009)	19,00	28,00	67,00
	metodologia proposta	**7,57**	8,58	**96,77**
Miyazaki94	(DOLADO, 2001)	-	50,60	47,90
	(WU; LI; LIANG, 2013)	24,00	32,40	51,90
	(ELISH; HELMY; HUSSAIN, 2013)	-	35,49	60,00
	metodologia proposta	**9,87**	**14,79**	**87,50**
Nasa	(BRAGA et al., 2007)	-	16,39	88,89
	(ELISH, 2009)	-	**5,51**	**94,44**
	(OLIVEIRA et al., 2010)	-	16,50	**94,44**
	metodologia proposta	**7,48**	12,02	87,10
Tukutuku	(MENDES, 2007)	27,40	34,30	33,30
	(CORAZZA et al., 2011)	41,00	59,00	34,00
	(CORAZZA et al., 2010)	39,70	72,30	39,55
	metodologia proposta	**11,49**	**26,75**	**75,90**

Fonte: Elaborada pelo autor

melhores resultados em comparação com outros trabalhos, a mesma pode fazer estimativas coerentes em todos os *datasets* de forma automática e generalista ao problema.

5.4 Ameaças a Validade

As ameaças a validade são fatores que podem influenciar os resultados, podendo até mesmo invalidar os mesmos (WOHLIN et al., 2012). Wohlin et al. (2012) dividem as ameaças a validade em quatro categorias. Validade de conclusão, que está relacionada ao relacionamento entre o tratamento dos experimentos e o resultado. Validade interna, esta ameaça está relacionada ao fato dos resultados observados serem causados pelo tratamento empregado ou por outro fator. Validade de construção, que considera os relacionamentos entre a teoria e a observação, isto é, se o tratamento reflete a causa e o resultado reflete o efeito. E por fim, a validade externa, que trata da generalização dos resultados dos experimentos.

5.4.1 Validade de Conclusão

A validade de conclusão diz respeito a habilidade de chegar a uma conclusão correta considerando os relacionamentos entre o tratamento e o resultado do experimento. Uma das ameaças presentes é a heterogeneidade dos *datasets*, para amenizar esta ameaça foi aplicado uma transformação logarítmica que ajudou a normalizar os *datasets* como apresentados nos resultados.

Existe ainda a ameaça em relação a confiabilidade em relação a medição dos resultados. Esta ameaça foi tratada usando métricas conhecidas para estimativa de esforço no desenvolvimento de software. O que permitiu analisar e também comparar os resultados gerados com outros da literatura.

5.4.2 Validade Interna

A validade interna diz respeito a relação de causalidade entre a metodologia proposta e os resultados aferidos nos experimentos. Dentre as principais ameaças a validade interna pode-se destacar o tamanho dos *datasets* usados nos experimentos. Para minimizar esta ameaça foram realizados experimentos usando quatro *designs* diferentes, onde a quantidade de dados usados na etapa de treinamento foi diferente de acordo com o design usado.

Outra ameaça aos experimentos foi a possibilidade de criação de subgrupos com os mesmos projetos para os experimentos com validação cruzada de 15, 10 e 5 *folds*. Essa ameça foi tratada realizando a criação dos subgrupos de forma aleatória, assim, seriam criados grupos diferentes a cada execução.

Existe ainda a ameaça em relação ao hardware usado para realização dos experimentos. Como a técnica de Auto-Sklearn é dependente do tempo escolhido pelo usuário, o hardware pode ser um fator limitante para realização de melhores estimativas. Para amenizar esta ameaça foi usado o tempo padrão de execução do Auto-Sklearn que é de 120 segundos.

5.4.3 Validade de Construção

A validade de construção esta relacionada as ameaças que afetam o design do experimento. Uma ameaça latente, é usar no treinamento dos AutoMLs características que fossem derivadas do valor de esforço, ou ainda, que não tivessem relação alguma com o esforço. Esta ameaça foi minimizada através da etapa de seleção de características, onde foram selecionadas as características que tivessem relação com o esforço de acordo com a literatura.

Outra ameaça esta relacionada com as métricas empregadas para avaliação da metodologia. Foram escolhidas métricas que são amplamente usadas para estimativa de

esforço (WEN et al., 2012; IDRI; HOSNI; ABRAN, 2016). Entretanto, a métrica MMRE não é uma unanimidade no que diz respeito a estimativa de esforço (TIERNO; NUNES, 2013), uma vez que é afetada por superestimações, entretanto, esta é uma das mais usadas na estimativa de esforço (IDRI; HOSNI; ABRAN, 2016). Para minimizar esta ameaça foram usadas outras métricas de estimativa que são menos influenciadas por superestimativas como é o caso da MdMRE (TIERNO; NUNES, 2013).

5.4.4 Validade Externa

As ameaças a validade externa diz respeito a capacidade de generalização dos resultados. Com relação a essas ameaças destaca-se primeiramente a quantidade de *datasets* usados para realização dos experimentos. Ainda em relação aos *datasets* usados, outra ameaça foi a diversidade das métricas usadas para coleta dos dados. Os *datasets* foram possuem projetos desenvolvidos seguindo quatro métricas, COCOMO, Pontos de Função, Pontos de Caso de Uso e uma própria de desenvolvimento de softwares Web. Buscou-se minimizar estas ameaças ao coletando a maior quantidade de *datasets* possíveis, fazendo buscas em repositórios de engenharia de software e também entrando em contato com pesquisadores.

Outra ameaça é o tempo de coleta dos *datasets*. A maioria dos *datasets* foram coletados mais de 10 anos. Os dados podem não terem uma representatividade em relação a como são desenvolvidos os softwares atualmente. Entretanto, Minku e Yao (2017) afirmaram que projetos antigos e atuais podem compartilhar semelhanças entre si, assim, embora os *datasets* tenham dados de projetos antigos, estes ainda podem ainda ser relevantes para estimativa de esforço na atualidade.

5.5 Considerações Finais

Os resultados expostos neste capítulo demonstram a aplicação da metodologia proposta para *datasets* de estimativa de esforço. Os experimentos realizados indicam que a metodologia é promissora considerando os resultados para as métricas MMRE e PRED(25). Além de auxiliar as estimativas a metodologia pode ainda implicar na melhoria da qualidade dos softwares ao auxiliar numa melhor alocação de recursos durante o desenvolvimento do software.

Entretanto, são necessários mais experimentos, principalmente com *datasets* mais recentes, e ainda a realização de testes com projetos reais a serem desenvolvidos. Apesar disso, os resultados iniciais mostram-se promissores.

O capítulo seguinte traz as conclusões e discussões gerais sobre este trabalho de mestrado, além de propor melhorias e trabalhos futuros.

6 Conclusões e Trabalhos Futuros

A estimativa de esforço é um dos problemas enfrentados por profissionais da indústria de desenvolvimento de software. Alguns métodos foram propostos ao longo dos últimos anos para auxiliar os profissionais na atividade de mensurar o esforço. As técnicas de AM têm ganhado destaque na estimativa de esforço por realizarem estimativas mais adequadas em relação ao esforço de desenvolvimento de software.

Este trabalho de mestrado se propõe a criar uma metodologia para estimativa de esforço no desenvolvimento de software empregando técnicas de AutoML. Foram escolhidas duas técnicas de AutoML, Auto-Sklearn e TPOT, para a metodologia proposta. Para avaliar as estimativas geradas pela metodologia foram usados 10 *datasets* de estimativa de esforço. Os resultados foram avaliados usando as métricas MMRE, PRED(25), MAE e MdMRE.

Os resultados apresentaram indícios que a metodologia proposta pode realizar estimativas adequadas considerando as métricas de MMRE e PRED(25). Nos experimentos que usaram a maior quantidade de dados possíveis para o treinamento (*Leave-One-Out*), ambos os AutoML mantiveram-se acima dos 75%. Entretanto, no *dataset* Tukutuku os resultados foram de 30,16% e 26,75% para o Auto-Sklearn e TPOT respectivamente. Embora os resultados não tenham ficado abaixo dos 25%, a comparação com outros trabalhos mostra que as estimativas para este *dataset* ainda estão próximas ao esforço real. Em contrapartida, no *dataset* Desharnais os resultados mantiveram-se nos 100% para a métrica de PRED(25) o que indica que todas os erros relativos para este *dataset* esteve sempre abaixo dos 25%.

Considerando os resultados dos experimentos pode-se ainda afirmar que ambas as técnicas avaliadas conseguiram realizar boas estimativas. Compreende-se ainda que os resultados levantam indícios que a metodologia proposta pode auxiliar pesquisadores e engenheiros de software a realizarem estimativas mais adequadas para o desenvolvimento de software. Portanto, se entende que os objetivos deste trabalho foram atingidos.

Como trabalhos futuros sugere-se a aplicação da metodologia apresentada noutros *datasets* a fim de obter-se mais robustez em relação às conclusões aqui apresentadas, sugere-se ainda um estudo de caso para avaliar a metodologia no auxílio de estimativas de um projeto a ser desenvolvido.

Sugere-se ainda a geração de dados sintéticos no treinamento das técnicas de AutoML, considerando que *datasets* de engenharia de software costumam ter poucos dados. Além disso é necessário estudos sobre a criação de uma ferramenta para estimativa de esforço na indústria destinada aos profissionais de desenvolvimento de software e que não

possuam um prévio conhecimento sobre AM.

Finalmente, a pesquisa conduzida durante esta dissertação resultou num artigo científico publicado no Simpósio Brasileiro de Qualidade de Software(SBQS) no ano de 2019, de título: *Using Machine Learning Technique for Effort Estimation in Software Development.*

Referências

ABDELLATIF, T. M. A comparison study between soft computing and statistical regression techniques for software effort estimation. In: IEEE. *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. [S.l.], 2018. p. 1–5. Citado na página 14.

ALAA, A. M. et al. Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 uk biobank participants. *PloS one*, Public Library of Science San Francisco, CA USA, v. 14, n. 5, p. e0213653, 2019. Citado na página 16.

ALAA, A. M.; SCHAAR, M. van der. Prognostication and risk factors for cystic fibrosis via automated machine learning. *Scientific reports*, Nature Publishing Group, v. 8, n. 1, p. 1–19, 2018. Citado na página 16.

ALBRECHT, A. J.; GAFFNEY, J. E. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, IEEE, n. 6, p. 639–648, 1983. Citado na página 24.

AMARAL, W. et al. Using machine learning technique for effort estimation in software development. In: *Proceedings of the XVIII Brazilian Symposium on Software Quality*. [S.l.: s.n.], 2019. p. 240–245. Citado 2 vezes nas páginas 31 e 56.

ANGELIS, L.; STAMELOS, I. A simulation tool for efficient analogy based cost estimation. *Empirical software engineering*, Springer, v. 5, n. 1, p. 35–68, 2000. Citado na página 56.

ARAUJO, R. d. A.; OLIVEIRA, A. L. de; SOARES, S. Hybrid intelligent design of morphological-rank-linear perceptrons for software development cost estimation. In: IEEE. *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*. [S.l.], 2010. v. 1, p. 160–167. Citado na página 56.

AZZEH, M.; NASSIF, A. B. Project productivity evaluation in early software effort estimation. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 30, n. 12, p. e2110, 2018. Citado na página 55.

AZZEH, M.; NASSIF, A. B.; MINKU, L. L. An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. *Journal of Systems and Software*, v. 103, p. 36 – 52, 2015. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121215000199>. Citado na página 36.

BALAJI, A.; ALLEN, A. Benchmarking automatic machine learning frameworks. *arXiv preprint arXiv:1808.06492*, 2018. Citado 5 vezes nas páginas 16, 17, 32, 38 e 52.

BANIMUSTAFA, A. Predicting software effort estimation using machine learning techniques. In: . [S.l.: s.n.], 2018. p. 249–256. Citado na página 31.

BARRY, B. et al. Software engineering economics. *New York*, v. 197, 1981. Citado na página 21.

BOEHM, B.; ABTS, C.; CHULANI, S. Software development cost estimation approaches—a survey. *Annals of software engineering*, Springer, v. 10, n. 1-4, p. 177–205, 2000. Citado na página 20.

BRAGA, P. L. et al. Bagging predictors for estimation of software project effort. In: IEEE. *2007 International Joint Conference on Neural Networks*. [S.l.], 2007. p. 1595–1600. Citado na página 56.

CARTWRIGHT, M. H.; SHEPPERD, M. J.; SONG, Q. Dealing with missing software project data. In: IEEE. *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)*. [S.l.], 2004. p. 154–165. Citado na página 40.

CHANDRA, A.; YAO, X. Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms*, Springer, v. 5, n. 4, p. 417–445, 2006. Citado na página 16.

CHARETTE, R. N. Why software fails [software failure]. *IEEE spectrum*, IEEE, v. 42, n. 9, p. 42–49, 2005. Citado 2 vezes nas páginas 14 e 15.

CHIU, N.-H.; HUANG, S.-J. The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software*, Elsevier, v. 80, n. 4, p. 628–640, 2007. Citado na página 56.

Chung, C. et al. Automated machine learning for internet of things. In: *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*. [S.l.: s.n.], 2017. p. 295–296. Citado na página 16.

CORAZZA, A. et al. How effective is tabu search to configure support vector regression for effort estimation? In: *Proceedings of the 6th international conference on predictive models in software engineering*. [S.l.: s.n.], 2010. p. 1–10. Citado na página 56.

CORAZZA, A. et al. Investigating the use of support vector regression for web effort estimation. *Empirical Software Engineering*, Springer, v. 16, n. 2, p. 211–243, 2011. Citado na página 56.

DALKEY, N. C. *Delphi*. [S.l.], 1967. Citado na página 20.

DOLADO, J. J. On the problem of the software cost function. *Information and Software Technology*, Elsevier, v. 43, n. 1, p. 61–72, 2001. Citado na página 56.

DRAGICEVIC, S.; CELAR, S.; TURIC, M. Bayesian network model for task effort estimation in agile software development. *Journal of systems and software*, Elsevier, v. 127, p. 109–119, 2017. Citado na página 19.

ELISH, M. O. Improved estimation of software project effort using multiple additive regression trees. *Expert Systems with Applications*, Elsevier, v. 36, n. 7, p. 10774–10778, 2009. Citado na página 56.

ELISH, M. O.; HELMY, T.; HUSSAIN, M. I. Empirical study of homogeneous and heterogeneous ensemble models for software development effort estimation. *Mathematical Problems in Engineering*, Hindawi, v. 2013, 2013. Citado na página 56.

FEURER, M. et al. Efficient and robust automated machine learning. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 2962–2970. Citado na página 33.

GRENNING, J. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, v. 3, p. 22–23, 2002. Citado na página 20.

IDRI, A.; HOSNI, M.; ABRAN, A. Systematic literature review of ensemble effort estimation. *Journal of Systems and Software*, Elsevier, v. 118, p. 151–175, 2016. Citado 13 vezes nas páginas 15, 16, 17, 29, 31, 32, 36, 43, 44, 53, 54, 55 e 58.

JODPIMAI, P.; SOPHATSATHIT, P.; LURSINSAP, C. Ensemble effort estimation using selection and genetic algorithms. *International Journal of Computer Applications in Technology*, Inderscience Publishers (IEL), v. 58, n. 1, p. 17–28, 2018. Citado 2 vezes nas páginas 16 e 37.

KARNER, G. Resource estimation for objectory projects. *Objective Systems SF AB*, Citeseer, v. 17, p. 1–9, 1993. Citado na página 26.

KHAZAIEPOOR, M.; BARDSIRI, A. K.; KEYNIA, F. A dataset-independent model for estimating software development effort using soft computing techniques. *Applied Computer Systems*, Sciendo, v. 24, n. 2, p. 82–93, 2019. Citado na página 56.

KITCHENHAM, B. A.; MENDES, E.; TRAVASSOS, G. H. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, IEEE, v. 33, n. 5, p. 316–329, 2007. Citado na página 54.

KUMAR, K. V. et al. Software development cost estimation using wavelet neural networks. *Journal of Systems and Software*, Elsevier, v. 81, n. 11, p. 1853–1867, 2008. Citado na página 56.

LI, Y.-F.; XIE, M.; GOH, T. A study of mutual information based feature selection for case based reasoning in software cost estimation. *Expert Systems with Applications*, Elsevier, v. 36, n. 3, p. 5921–5931, 2009. Citado na página 56.

LIRA, W. A. L. et al. Estimativa de esforço em projetos Ágeis de software utilizando mapas de kohonen. In: SBQS. *XIV Simpósio Brasileiro De Qualidade De Software*. [S.l.], 2015. Citado na página 14.

LOPES, M. d. O. B. A. Programação genética aplicada a estimativas de projeto de software. In: WESB. *X Workshop de Engenharia de Software Baseada em Buscas*. [S.l.], 2019. p. 1–2. Citado na página 56.

MARSLAND, S. *Machine learning: an algorithmic perspective*. [S.l.]: CRC press, 2015. Citado 4 vezes nas páginas 16, 30, 31 e 41.

MENDES, E. A comparison of techniques for web effort estimation. In: IEEE. *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. [S.l.], 2007. p. 334–343. Citado na página 56.

MENDES, E. *Practitioner's Knowledge Representation*. [S.l.]: Springer, 2014. Citado 3 vezes nas páginas 19, 21 e 28.

MENDES, E. et al. Effort estimation: how valuable is it for a web company to use a cross-company data set, compared to using its own single-company data set? In: *Proceedings of the 16th international conference on World Wide Web*. [S.l.: s.n.], 2007. p. 963–972. Citado 3 vezes nas páginas 39, 40 e 44.

MENDES, E.; MOSLEY, N.; WATSON, I. A comparison of case-based reasoning approaches. In: ACM. *Proceedings of the 11th international conference on World Wide Web*. [S.l.], 2002. p. 272–280. Citado 5 vezes nas páginas 15, 20, 21, 29 e 36.

MENZIES, T.; GREENWALD, J.; FRANK, A. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, IEEE, v. 33, n. 1, p. 2–13, 2006. Citado na página 40.

MINKU, L. L.; YAO, X. Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models. *Automated Software Engineering*, Springer, v. 24, n. 3, p. 499–542, 2017. Citado na página 58.

NAGPAL, G.; UDDIN, M.; KAUR, A. A comparative study of estimation by analogy using data mining techniques. *JIPS*, v. 8, n. 4, p. 621–652, 2012. Citado na página 56.

NASSIF, A. B. et al. A comparison between decision trees and decision tree forest models for software development effort estimation. In: IEEE. *2013 Third International Conference on Communications and Information Technology (ICCIT)*. [S.l.], 2013. p. 220–224. Citado na página 56.

OLIVEIRA, A. L. et al. Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology*, Elsevier, v. 52, n. 11, p. 1155–1166, 2010. Citado na página 56.

OLSON, R. S. et al. Applications of evolutionary computation: 19th european conference, evoapplications 2016, porto, portugal, march 30 – april 1, 2016, proceedings, part i. In: _____. Springer International Publishing, 2016. cap. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, p. 123–137. ISBN 978-3-319-31204-0. Disponível em: <http://dx.doi.org/10.1007/978-3-319-31204-0_9>. Citado na página 34.

POSPIESZNY, P.; CZARNACKA-CHROBOT, B.; KOBYLINSKI, A. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*, Elsevier, v. 137, p. 184–196, 2018. Citado 3 vezes nas páginas 14, 36 e 37.

SATAPATHY, S. M.; ACHARYA, B. P.; RATH, S. K. Early stage software effort estimation using random forest technique based on use case points. *IET Software*, IET, v. 10, n. 1, p. 10–17, 2016. Citado 2 vezes nas páginas 15 e 37.

SHAH, M. A. et al. Minn: A missing data imputation technique for analogy-based effort estimation. *International Journal of Advanced Computer Science and Applications*, v. 10, 01 2019. Citado na página 40.

SHEPPERD, M.; KADODA, G. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, IEEE, v. 27, n. 11, p. 1014–1022, 2001. Citado na página 36.

SHEPPERD, M.; MACDONELL, S. Evaluating prediction systems in software project estimation. *Information and Software Technology*, Elsevier, v. 54, n. 8, p. 820–827, 2012. Citado na página 30.

SHIRABAD, J. S.; MENZIES, T. *The PROMISE Repository of Software Engineering Databases*. 2005. School of Information Technology and Engineering, University of Ottawa, Canada. Disponível em: <http://promise.site.uottawa.ca/SERepository>. Citado na página 44.

SILHAVY, R.; SILHAVY, P.; PROKOPOVA, Z. Algorithmic optimisation method for improving use case points estimation. *PloS one*, Public Library of Science, v. 10, n. 11, p. e0141887, 2015. Citado na página 55.

SOFTWARE, A. B. das Empresas de. *Mercado Brasileiro de Software: panorama e tendências*. [S.l.]: ABES-Associação Brasileira das Empresas de Software São Paulo, 2019. Citado na página 14.

SONG, L.; MINKU, L. L.; YAO, X. The impact of parameter tuning on software effort estimation using learning machines. In: *Proceedings of the 9th international conference on predictive models in software engineering*. [S.l.: s.n.], 2013. p. 1–10. Citado na página 16.

SONG, L.; MINKU, L. L.; YAO, X. A novel automated approach for software effort estimation based on data augmentation. In: ACM. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. [S.l.], 2018. p. 468–479. Citado 8 vezes nas páginas 14, 19, 29, 36, 37, 40, 41 e 55.

Sá, A. D.; PAPPA, G. A hyper-heuristic evolutionary algorithm for learning bayesian network classifiers. In: . [S.l.: s.n.], 2014. v. 8864, p. 430–442. Citado na página 33.

TANAKA, K.; MONDEN, A.; YÜCEL, Z. Prediction of software defects using automated machine learning. In: IEEE. *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. [S.l.], 2019. p. 490–494. Citado 2 vezes nas páginas 17 e 38.

THORNTON, C. et al. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2013. p. 847–855. Citado na página 33.

TIERNO, I. A.; NUNES, D. J. An extended assessment of data-driven bayesian networks in software effort prediction. In: IEEE. *2013 27th Brazilian Symposium on Software Engineering*. [S.l.], 2013. p. 157–166. Citado 6 vezes nas páginas 29, 30, 40, 42, 54 e 58.

WAZLAWICK, R. *Engenharia de software: conceitos e práticas*. [S.l.]: Elsevier Editora Ltda., 2013. Citado 2 vezes nas páginas 15 e 40.

WEN, J. et al. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, Elsevier, v. 54, n. 1, p. 41–59, 2012. Citado 10 vezes nas páginas 14, 15, 19, 28, 29, 31, 36, 43, 44 e 58.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012. Citado 2 vezes nas páginas 19 e 56.

WU, D.; LI, J.; LIANG, Y. Linear combination of multiple case-based reasoning with optimized weight for software effort estimation. *The Journal of Supercomputing*, Springer, v. 64, n. 3, p. 898–918, 2013. Citado na página 56.