Estimativa de Esforço em Projetos de Software com Técnicas de Aprendizado de Máquina*

Carlos A. Bitencourt[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
79070-900 – Campo Grande – MS – Brazil

carlos.bitencourt@ufms.br

Abstract. *The need to develop reliable and secure digital solutions and services leveraged the quest to improve Software Engineering activities. Among them, Software Effort Estimation has been improved with the use of Machine Learning. Thus, this article proposes to evaluate the accuracy of five Machine Learning Algorithms in five public domain databases, showing a better result in the DT, RF and SVM algorithms respectively.*

Resumo. *A necessidade do desenvolvimento de soluções e serviços digitais confiáveis e seguros alavancaram a busca pela melhoria de atividades de Engenharia de Software. Dentre elas a Estimativa de Esforço de Software vem sendo aprimorada com o uso de Aprendizado de Máquina. Deste modo, este artigo propõe-se a avaliar a acurácia de cinco Algoritmos de Aprendizado de Máquina em cinco bases de dados de domínio público constantando um melhor resultado nos algoritmos DT, RF e SVM respectivamente.*

1. Introdução

No momento atual, é notório a necessidade do desenvolvimento de soluções e serviços digitais confiáveis e seguros. Para alcançar este objetivo, a Engenharia de Software é essencial para a entrega de produtos e serviços com qualidade. Uma das atividades mais importantes é a **Estimativa de Esforço de Software - EES**. A EES, pode ser aprimorada com o tempo conforme experiência da equipe. Uma vez que a estimativa pode ser aferida por meio de dados históricos, seria possível a implementação de uma solução de **Aprendizado de Máquina (AM)** para a realização de análise das métricas de um projeto de software? Assim sendo, poder-se-ia aproveitar a base de conhecimento de outras equipes para a predição de estimativas? Reduzindo assim análises equivocadas e tornando cada vez mais preciso as estimativas dos cronogramas dos projetos.

Embora muitas pesquisas tenham sido feitas em EES nos últimos anos, a precisão das estimativas ainda é uma grande preocupação, tendo em vista, que estimativas imprecisas podem levar a atrasos e aumento de custos em um projeto. Nas últimas décadas, foram propostos vários métodos para estimativa de esforço de software os quais, normalmente, descrevem a estimativa de esforço por meio de fórmulas que consideram parâmetros históricos como por exemplo, experiência da equipe, linguagem de programação dentre outros [Ali e Gravino 2019].

Assim sendo, o objetivo desse trabalho é realizar uma exploração preliminar da aplicação de ao menos cinco algoritmos de Aprendizagem de Máquina sobre algumas bases de dados públicas contendo informações de EES.

*Dados complementares, código, datasets: https://github.com/carlos-bitencourt/ia-projects

2. Referencial Teórico

Na fundamentação teórica, para que se possa embasar uma análise adequada, faz-se necessário enumerar e descrever os modelos e técnicas de aprendizado de máquina as serem utilizados nesta pesquisa, bem como uma breve análise das pesquisas sobre estimativa de esforço de software com Aprendizado de Máquina.

2.1. Aprendizado de Máquina (AM)

Um dos primeiros exemplos de um sistema de aprendizagem de máquina, foi um programa desenvolvido por Samuel (1959) para jogar damas. Com este programa, foi possível demonstrar como um programa poderia melhorar o seu processamento de dados, sem ser programado. Ou seja, um programa poderia ser treinado para fazer previsões [IEEE-CS 2013, Samuel 1959].

Ao longo dos anos, por meio do desenvolvimento de técnicas, métodos e novos algoritmos para o campo da AM foi possível desenvolver uma série de serviços [Abadal et al. 2020].

Conceitualmente pode-se delinear a AM com o postulado de Mitchell (1997):

> Um programa de computador é dito aprender a partir de uma experiência E com respeito a uma classe de tarefas T e medida de desempenho P, se seu desempenho em tarefas de T, medido por P, melhora a experiência E. [Mitchell 1997, p. 2, tradução nossa]

Deste modo, por meio de modelos que representam hipóteses, é possível medir o conhecimento aprendido por um processo de indução, que pode ser visto como um processo de busca em que se almeja encontrar a melhor hipótese [Russell e Norvig 2013].

Nas subseções a seguir serão apresentados alguns técnicas, conceitos e algoritmos de aprendizagem de máquina que serão utilizados nesta pesquisa.

2.1.1. Árvores de Decisão (Decision Tree - DT)

Uma árvores de decisão toma como entrada um vetor de atributos e retorna uma decisão única. Os valores de entrada e saída podem ser discretos ou contínuos.

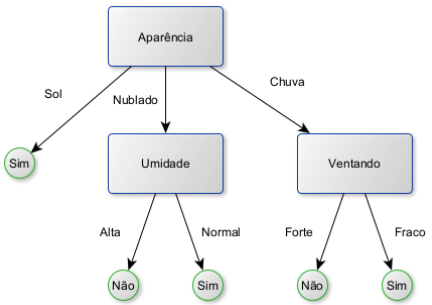

Figura 1. Árvore de Decisão para jogar tênis. Adaptado de Mitchell (1997).

Na Figura 1, pode-se observar um exemplo de árvore de decisão, que representa de forma natural ao ser humano os passos para decidir se deve ou não jogar tênis. Cada nó folha da árvore especifica o valor a ser retornado pela função.

Uma forma de medir a importância (ganho de informação) de uma atributo, é por meio do conceito de **Entropia**, que é a medida de incerteza de uma variável aleatória. Quanto mais informações são necessárias para aprender, maior é a entropia, por outro lado, se não é necessário nenhuma informação a entropia é igual a zero. Dado que a entropia mensura o quão impuro os dados de um coleção estão, pode-se definir o ganho de informação de um atributo A em relação a uma coleção V como:

$$Ganho(V, A) = Entropia(V) - \sum_{v \in A_k} \frac{|V_v|}{|V|} Entropia(V_v) \tag{1}$$

Assim sendo, é possível calcular o Ganho de informação para realização do treinamento. Para dados contínuos, como por exemplo previsão de preços, o algoritmo de árvore de decisão deve encontrar pontos de divisão entre os valores [Mitchell 1997, Russell e Norvig 2013].

2.1.2. Redes Neurais Artificiais (RNA)

Inspirado no funcionamento do cérebro humano, as Redes Neurais Artificiais são representadas por uma coleção de classificadores lineares conectados cujas propriedades são determinadas pela topologia e pelas propriedades dos "neurônios" (Figura 2).

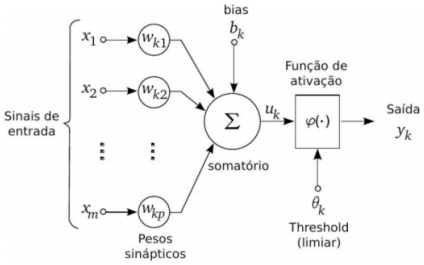

Figura 2. Modelo de Neurônio. Fonte: [Haykin 2001]

Um neurônio é uma unidade de processamento de informação onde podemos identificar três elementos básicos. O primeiro é um conjunto de sinais de entradas com *sinapses* caracterizada por pesos ou forças. O segundo, um somador que trata os sinais de entrada. E, por fim uma função de ativação que limita os sinais de saída. Em termos matemáticos pode-se descrever um neurônio k com os sequintes pares de equações:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2}$$

e

$$y_k = \varphi(u_k + b_k) \tag{3}$$

de modo que $x_1, x_2, \ldots, x_n$ são os sinais de entrada; $w_{k1}, w_{k2}, \ldots, w_{km}$ são os pesos sinápticos; u_k a saída dos sinais de entrada; e y_k é o sinal de saída.

É possível organizar a topologia de uma rede neural em diversas camadas, com diferentes número de unidades por camada, aumentando a capacidade da rede de aprender [Haykin 2001].

2.1.3. K-Vizinhos Mais Próximos (K-Nearest Neighbors - KNN)

Algoritmos baseados em instâncias baseiam o seu aprendizado no armazenamento dos dados de treinamento para utilização na classificação de uma entrada nova. Um dos algoritmos mais utilizados nesta abordagem é o **KNN**.

Os vizinhos mais próximos podem ser calculados pela **distância euclidiana**. Dado uma instância x descrita pelo vetor $[a_1(x), a_2(x), \ldots a_n(x)]$, onde a_k é o k-ésimo atributo da instância x, então a distância de x_i e x_j é definida:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (a_k(x_i) + a_k(x_j))^2} \tag{4}$$

Para classificar, primeiro encontre os k vizinhos mais próximos e realize um processo de contagem. A classe majoritária será atribuída ao novo objeto. Para evitar empates, recomenda-se sempre escolher um número ímpar para k. Para classes contínuas, pode-se tirar a média ou mediana de k vizinhos [Mitchell 1997, Russell e Norvig 2013].

2.1.4. Máquinas de Vetor de Suporte (Support Vector Machine - SVM)

A **Máquina de Vetor de Suporte (SVM)** contém três propriedades relevantes. A primeira, diz respeito a capacidade de construir um **separador de margem máxima** que contribui na generalização. A segunda, é a capacidade de, além de tratar dados lineares, incorporar os dados em um espaço de dimensão superior. Terceiro, apesar de ser um método não paramétrico, na prática, acabam mantendo apenas uma fração do número de exemplos. A idéia é considerar alguns exemplos mais importantes que os outros o que pode levar a uma generalização melhor.

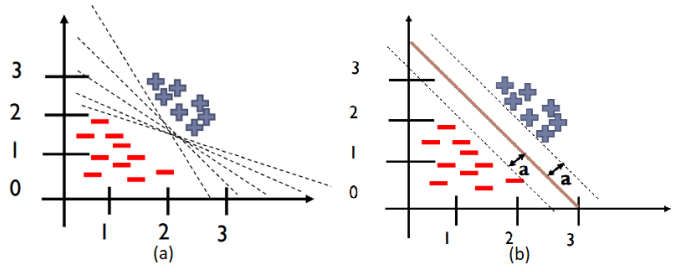

Figura 3. Máquina de Vetor de Suporte. (a) Duas classes vários separadores lineares candidatos. (b) O separador de margem máxima ao centro. Adaptado de [Nogueira 2020]

Na Figura 3, pode-se observar em (a) várias possibilidades de separação, enquanto em (b) busca-se o separador médio mais distantes das classes, chama-se o separador de **margem máxima**, dessa forma pode-se minimizar a perda de generalização [Russell e Norvig 2013].

2.1.5. Floresta Aleatória (Random Florest - RF)

O Algoritmo de Floresta Aleatória consiste em um classificador cuja topologia é uma coleção de classificadores estruturados em árvores.

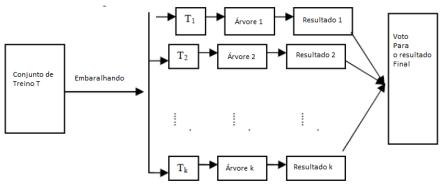

Figura 4. Modelo de Floresta Aleatória. Adaptado de [Liu et al. 2012]

Como pode-se observar na Figura 4, cada árvore recebe um conjunto de treinamento aleatório, que após os dados toma uma decisão representada pela equação 5:

$$H(x) = arg\max_{Y} \sum_{i=1}^{k} I(h_i(x) = Y) \tag{5}$$

onde $H(x)$ é uma combinação do modelo de classificação, h_i é a decisão de uma das árvores, Y é a variável de saída. Cada árvore tem direito a um voto para selecionar o melhor resultado de classificação [Liu et al. 2012].

2.2. Estimativa de Esforço de Software e Aprendizagem de Máquina

Muitas pesquisas tem sido feitas na área de estimativas de esforço de softwares nos últimos anos. No entanto, a precisão das estimativas de esforço de software ainda é uma grande preocupação, tendo em vista, que estimativas imprecisas podem levar a atrasos e aumento de custos em um projeto. Nas últimas décadas, foram propostos vários métodos para estimativa de esforço de software os quais, normalmente, descrevem a estimativa de esforço por meio de fórmulas que consideram parâmetros históricos como por exemplo, experiência da equipe, linguagem de programação dentre outros [Ali e Gravino 2019].

Nas ultimas três décadas os estudos que aplicam o AM na estimativa de software foi ampliado motivado pela ampla margem de erro nos métodos tradicionais de estimativa e pela melhoria contínua dos algoritmos de aprendizado de máquina [BaniMustafa 2018].

Vários algoritmos foram utilizados para estimar esforço, custo e tempo de desenvolvimento de software. A maioria dessas técnicas foram aplicadas a dados públicos. Árvore de Decisão, Florestas de Árvores de Decisão em [Nassif et al. 2013], o uso de programação genética foi relatado em [Chavoya et al. 2012]. Uso de Redes Neurais Artificiais foram usadas para prever o esforço e foram comparadas com a estimativa do modelo COMOMO [de Barcelos Tronto et al. 2007, Bhatia e Attri 2015] dentre muitos outros trabalhos.

Ali e Gravino (2019) por meio de uma revisão sistemática da literatura sobre estimativa de eforço de software baseado em aprendizam de máquina, resgataram a análise sistemática de [Wen et al. 2012] e complementaram com os próximos anos. Nesta análise, eles reponderam questões como quais as bases de dados mais utilizadas nas pesquisas?

Quais os algoritmos quais obtiveram o maior desempenho? Quais as métricas mais utilizadas? Dentre os estudos avaliados, as três técnicas mais utilizadas foram respectivamente Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Redes Bayseanas. A técnica K-Vizinhos Mais Próximos em quinto, Árvores de Decisão em sexto e por fim, Floresta Aletória em oitava posição. Dentre as bases de dados mais usadas nos estudos, as três primeiras foram a NASA, COCOMO e ISBSG respectivamente.

Em relação as métricas, Magnitude Média do Erro Relativo (Mean Magnitude of Relative Error - MMRE) e Porcentagem de Previsões de 25 por cento (Percentage of Predictions - Pred(25)) foram amplamente usadas sendo que o Erro Absoluto Médio (Mean Absolute Error - MAE) ficou em quarto mais utilizado.

3. Metodologia

Para realização da pesquisa, foram selecionadas cinco bases de dados disponíveis publicamente, todas relacionadas a Estimativas de Esforço de Software (Tabela 1): Cocomo81 (D1), Cocomonasa (D2), Desharnais (D3), Nasanumeric (D4) e Seera (D5) [Sayyad Shirabad e Menzies 2005, Vanschoren et al. 2014, Mustafa e Osman 2020].

Tabela 1. Base de Dados

ID	Nº Atr. Inicial	Nº Atr. Final	Nº Linhas Inicial	Nº Linhas Final
Cocomo81 (D1)	17	17	63	63
Cocomonasa (D2)	17	17	60	60
Desharnais (D3)	12	17	81	60
Nasanumeric (D4)	24	41	93	93
Seera (D5)	76	33	120	111

As bases de dados D1, D2, D3 e D4 possuem atributos semelhantes relacionados a técnica de Estimativa de Software COCOMO [Boehm 1981]. Este modelo, estima o esforço de desenvolvimento de software em relação a homem-mês considerando o tamanho do código (LOC) ou Pontos por função [BaniMustafa 2018]. A fórmula geral:

$$Effort = AS^b \tag{6}$$

Sendo A um parâmetro de produtividade, b um parâmetro escalável de economia e S linhas por código (LOC) ou Pontos por função [Shepperd e Schofield 1997].

Deste modo, "Effort" representa o atributo Classe das Bases de Dados. A base de dados D5, possui dados mais recentes e completos contendo 76 atributos inicialmente, muito mais que as demais bases, 24 (D4), 12 (D3) e 17 (D1, D2) (Tabela 1).

Para tratamento dos dados, optou-se por excluir linhas cujos os dados fossem nulos ou vazios. Deste modo, após a identificação e exclusão destas linhas, não houve alteração nas bases D1, D2 e D3, foram removidas 19 linhas da base D4 e 11 linhas da base D5. Em relação aos atributos, foi necessário o ajuste de atributos textuais em numéricos nas bases D2 e D4, ajustes em atributos categóricos nas bases D3 ("language"), D4 ("cat2") e D5 ("Programming language used"), resultando na alteração do número dos

atributos finais como pode ser observado na Tabela 1. Além disso, foram removidos alguns atributos descritivos como ano de projeto e identificadores. Para a base D5, foram removidos vários atributos cujo valor compunha outro atributo presente, ou seja, dados derivados, mantendo-se atributos considerados relavantes e semelhantes as demais bases.

Após, as bases de dados foram submetidas a transformação na mesma ordem de grandeza por meio do método *z-score*.

3.1. Ajustes e Treinamento dos Classificadores / Regressores

Foram selecionados cinco Algoritmos para análise de performance (Tabela 2): Árvore de Decisão (DT), Redes Neurais Artificiais (Perceptron Multi-camadas - MLP), K-Vizinhos Mais Próximos (KNN), Máquinas de Vetor de Suporte (SVM) e Floresta Aleatório (RF).

Para implementação e execução da análise foi utilizado a biblioteca *Scikit-learn* [Pedregosa et al. 2011] com a linguagem de programação *Python*.

Tabela 2. Algoritmos e componentes *Scikit-learn*

Algoritmos	**Componentes**
DT	DecisionTreeRegressor
MLP	MLPRegressor
KNN	KNeighborsRegressor
SVM	SVR
RF	RandomForestRegressor

Para o treinamento optou-se por usar a validação cruzada de *3-folds* e a **Média do Erro Absoluto (MAE)** como métrica de avaliação da Acurária juntamente como Desvio Padrão. Foi adotado o MAE inverso e portanto interpreta-se os valores quanto maior melhor (Tabela 4).

Para cada algoritmo buscou-se a melhor configuração de parâmetros retornada pelo *GridSearchCV* por meio dos seguintes passos:

1. Para uma base e algoritmo, roda o *GridSearchCV* com opções de variáveis escolhidas.
2. Caso, o "melhor parâmetro" para uma determinada variável pertença a fronteira.
   (a) Ajusta as opções do parâmetro extendendo a fronteira. Ex: opções iniciais [2,4,8], suponha que o resultado, de melhor parâmetro, após a execução, seja [2], então ajusta-se os parâmetros para novas opções [0, 2, 4].
3. Volta ao Passo 1, até que os melhores valores de parâmetros não pertençam a fronteira de opções.

Deste modo, busca-se garantir que os melhores valores dos parâmetros não econtram-se fora dos limites testados.

4. Análise e Discussão

Os melhores parâmetros encontrados seguindo os passos descritos na Seção 3.1 para os cinco algoritmos propostos considerando a base de dados são apresentados na Tabela 3.

Tabela 3. Algoritmos, base de dados, melhores parâmetros

Algoritmos	Base de Dados	Melhores Parâmetros
DT	D1	criterion: absolute_error,max_depth: 8, splitter: random
KNN	D1	algorithm: ball_tree, n_neighbors: 4, weights: distance
MLP	D1	activation: relu, hidden_layer_sizes: 60, solver: lbfgs
RF	D1	max_depth: 6, n_estimators: 15
SVM	D1	C: 10, epsilon: 0.023, kernel: linear
DT	D2	criterion: absolute_error, max_depth: 9, splitter: random
KNN	D2	algorithm: auto, n_neighbors: 8, weights: distance
MLP	D2	activation: tanh, hidden_layer_sizes: 50, solver: lbfgs
RF	D2	max_depth: 10, n_estimators: 5
SVM	D2	C: 20, epsilon: 0.005, kernel: poly
DT	D3	criterion: squared_error, max_depth: 11,splitter: random
KNN	D3	algorithm: auto, n_neighbors: 4, weights: distance
MLP	D3	activation: relu, hidden_layer_sizes: 50, solver: sgd
RF	D3	max_depth: 8, n_estimators: 4
SVM	D3	C: 0.2, epsilon: 0.0005, kernel: sigmoid
DT	D4	criterion: absolute_error, max_depth: 10, splitter: random
KNN	D4	algorithm: auto, n_neighbors: 23, weights: distance
MLP	D4	activation: logistic, hidden_layer_sizes: 5, solver: sgd
RF	D4	max_depth: 11, n_estimators: 5
SVM	D4	C: 6, epsilon: 0.0002, kernel: poly
DT	D5	criterion: absolute_error, max_depth: 4, splitter: random
KNN	D5	algorithm: kd_tree, n_neighbors: 7, weights: distance
MLP	D5	activation: tanh, hidden_layer_sizes: 18, solver: adam
RF	D5	max_depth: 7, n_estimators: 7
SVM	D5	C: 4, epsilon: 0.13999, kernel: linear

Para os melhores parâmetros, os algoritmos alcançaram os desempenhos/acurácia relacionados na Tabela 4.

Analisando as informações relacionadas na Tabela 4, o algoritmo com maior Acurácia Média em relação as bases de dados foi o DT(0,691) seguido do RF (0,679) e SVM (0,669) respectivamente. Sendo que o DT teve maior desempenho na base D2 com 0,798 de Acurária e o pior na D3 com 0,513. A maior acurária em relação a um Algoritmo x Base foi o RF x D2 com 0,808. E, o pior desempenho DT x D3 com 0,513.

A Base de Dados a qual os algoritmos tiveram melhor desempenho foi a D2 com 0,751 de Acurácia Média seguida pela D1(0,699) e D4(0,680) respectivamente.

Deste modo, por meio dos dados apresentados, pode-se concluir que o Algoritmo que obteve melhor desempenho foi o DT seguido do RF e do SVM.

5. Conclusão

Durante a elaboração da pesquisa, constatou-se que existem vários trabalhos na área de estimativa de esforço de software com Aprendizado de Máquina. A maioria dessas

Tabela 4. Algoritmos e Desempenho em Erro Absoluto Médio (MAE) e Desvio Padrão. Inverso do Erro Absoluto Médio, lê-se quanto maior melhor.

Algoritmos	D1	D2	D3	D4	D5	Média
DT	0.753 $\pm$0.101	0.798 $\pm$0.013	0.513 $\pm$0.106	0.716 $\pm$0.047	0.679 $\pm$0.037	**0,691**
KNN	0.682 $\pm$0.194	0.607 $\pm$0.133	0.452 $\pm$0.070	0.672 $\pm$0.064	0.550 $\pm$0.094	**0,592**
MLP	0.656 $\pm$0.138	0.783 $\pm$0.055	0.551 $\pm$0.032	0.600 $\pm$0.054	0.591 $\pm$0.018	**0,636**
RF	0.709 $\pm$0.142	0.808 $\pm$0.047	0.477 $\pm$0.029	0.748 $\pm$0.041	0.655 $\pm$0.012	**0,679**
SVM	0.696 $\pm$0.140	0.759 $\pm$0.026	0.558 $\pm$0.142	0.666 $\pm$0.062	0.666 $\pm$0.034	**0,669**
Média	**0.699**	**0.751**	**0.510**	**0.680**	**0.628**	

técnicas foram aplicadas a dados de domínio público. Alguns algoritmos como Árvore de Decisão, Florestas de Árvores de Decisão foram relatados em [Nassif et al. 2013], o uso de programação genética foi relatado em [Chavoya et al. 2012]. Uso de Redes Neurais Artificiais foram usadas para prever o esforço e foram comparadas com a estimativa do modelo COMOMO [de Barcelos Tronto et al. 2007, Bhatia e Attri 2015] dentre outros trabalhos.

No entanto, apesar da existência dos trabalhos relatados, a escassez de base dados foi notória durante a realização da pesquisa. Muitas com poucos dados e dados de projetos antigos.

Cinco base de dados disponíveis em domínio público foram escolhidas (Tabela 1): Cocomonasa (D2), Desharnais (D3), Nasanumeric (D4) e Seera (D5) [Sayyad Shirabad e Menzies 2005, Vanschoren et al. 2014, Mustafa e Osman 2020]. Os algoritmos utilizados para treinamento foram (Tabela 2): Árvore de Decisão (DT), Redes Neurais Artificiais (Perceptron Multi-camadas - MLP), K-Vizinhos Mais Próximos (KNN), Máquinas de Vetor de Suporte (SVM) e Floresta Aleatório (RF).

Deste modo, concluisse, após os ajustes da base de dados e treinamento dos algoritmos, que os algoritmos com os melhores desempenhos por ordem de classificação foram: DT, RF e SVM. Além disso, este estudo pode servir de apoio para pesquisas na área de Estimativa de Software com apoio da Aprendizagem de Máquina para auxiliar na fundamentação das relações produzidas na pesquisa.

Referências

Abadal, S., Jain, A., Guirado, R., López-Alonso, J., e Alarcón, E. (2020). Computing graph neural networks: A survey from algorithms to accelerators. *CoRR*, abs/2010.00130.

Ali, A. e Gravino, C. (2019). A systematic literature review of software effort prediction using machine learning methods. *Journal of Software: Evolution and Process*, 31(10):e2211.

BaniMustafa, A. (2018). Predicting software effort estimation using machine learning techniques. In *2018 8th International Conference on Computer Science and Information*

Technology (CSIT), pages 249–256.

Bhatia, S. e Attri, V. K. (2015). Machine learning techniques in software effort estimation using cocomo datatset. *IJRDO - Journal of Computer Science Engineering*, 1(6):101–106.

Boehm, B. (1981). *Software Engineering Economics*. Prentice Hall.

Chavoya, A., Lopez-Martin, C., Andalon-Garcia, I. R., e Meda-Campaña, M. E. (2012). Genetic programming as alternative for predicting development effort of individual software projects. *PLOS ONE*, 7(11):1–10.

de Barcelos Tronto, I. F., da Silva, J. D. S., e Sant'Anna, N. (2007). Comparison of artificial neural network and regression models in software effort estimation. In *2007 International Joint Conference on Neural Networks*, pages 771–776.

Haykin, S. (2001). *Redes Neurais: princípio e prática*. Bookman.

IEEE-CS (2013). Computer pioneers by j. a. n. lee. Disponível em: `https://history.computer.org/pioneers/samuel.html`. Acessado em: Junho de 2022.

Liu, Y., Wang, Y., e Zhang, J. (2012). New machine learning algorithm: Random forest. In *Information Computing and Applications*, pages 246–252, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mitchell, T. (1997). *Machine learning*. McGraw-hill, New York, 1ª edição.

Mustafa, E. I. e Osman, R. (2020). The seera software cost estimation dataset.

Nassif, A. B., Azzeh, M., Capretz, L. F., e Ho, D. (2013). A comparison between decision trees and decision tree forest models for software development effort estimation. In *2013 Third International Conference on Communications and Information Technology (ICCIT)*, pages 220–224.

Nogueira, B. (2020). Support vector machine. Disponível em: `https://www.youtube.com/watch?v=eFaIhuCjkRU`. Acessado em: Junho de 2022.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., e Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Russell, S. e Norvig, P. (2013). *Inteligência Artificial*. Elsevier, Rio de Janeiro, 3ª edição.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Sayyad Shirabad, J. e Menzies, T. (2005). The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada.

Shepperd, M. e Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743.

Vanschoren, J., van Rijn, J. N., Bischl, B., e Torgo, L. (2014). Openml: networked science in machine learning. *CoRR*, abs/1407.7722.

Wen, J., Li, S., Lin, Z., Hu, Y., e Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59.