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. *Write after. Write after.Write*

Resumo. *Escrever depois, máximo 10 linhas. Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois,Escrever depois.*

1. Introdução

No momento atual, é notório a necessidade do desenvolvimento de soluções e serviços digitais confiáveis e seguros. Para alcançar este objetivo, a melhoria do Processo de Desenvolvimento de Software (PDS) é essencial para a entrega de produtos e serviços com qualidade. Uma das atividades mais importantes do PDS é a **estimativa de esforço de software**. A estimativa de software, 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 (AP)** 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 na área de estimativas de esforço de softwares nos últimos anos, 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].

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 estimativa de esforço de sotware.

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 do estado da arte das pesquisas sobre estimativa de esforço de software com Aprendizado de Máquina.

2.1. Aprendizado de Máquina (AP)

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 AP foi possível desenvolver uma série de serviços [Abadal et al. 2020].

Conceitualmente pode-se delinear a AP 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

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. Cada nó interno da árvore corresponde a um teste do valor de um dos atributos de entrada A_i, e as ramificações dos nós são classificadas com os valores possíveis do atributo $A_i = Valor_{ik}$.

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. O Algoritmo de Árvore de Decisão adota uma estratégia "gulosa" de dividir para conquista. Sempre testar o atributo mais relevante primeiro pode diminuir aumentar as chances de alcançar a classificação correta com um número reduzido de testes.

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. Dado um problema de classificação booleana, a entropia relativa de um subconjunto V,

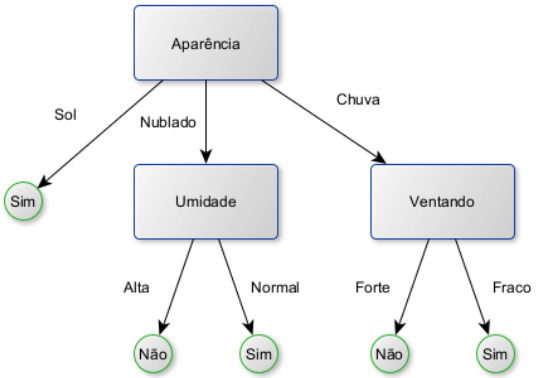

Figura 1. Árvore de Decisão para jogar tênis. Adaptado de Mitchell (1997).

com k diferentes valores v_k, cada uma com probabilidade $P(v_k)$, é definida como:

$$Entropia(V) = -\sum_{k} P(v_k) \log_2 P(v_k) \tag{1}$$

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:

$$Ganho(V, A) = Entropia(V) - \sum_{v \in A_k} \frac{|V_v|}{|V|} Entropia(V_v) \tag{2}$$

Assim sendo, é possível calcular o Ganho de informação para realização do treinamento. Outra questão relevante, é que nem sempre os dados são discretos, existem também dados contínuos, como por exemplo previsão de preços. Nestes casos 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 é representada por uma coleção de classificadores lineares conectados cujas propriedades são determinadas pela topologia e pelas propriedades dos "neurônios" (Figura 2).

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:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3}$$

e

$$y_k = \varphi(u_k + b_k) \tag{4}$$

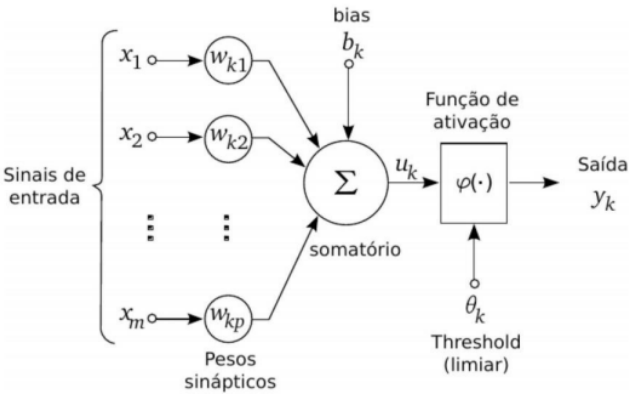

Figura 2. Modelo de Neurônio. Fonte: [Haykin 2001]

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)

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 aboradagem é o **K-Vizinhos Mais Próximos (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:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} (a_k(x_i) + a_k(x_j))^2} \tag{5}$$

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)

A **Máquina de Vetor de Suporte (SVM)** é uma abordagem popular da aprendizagem supervisionada.

O 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.

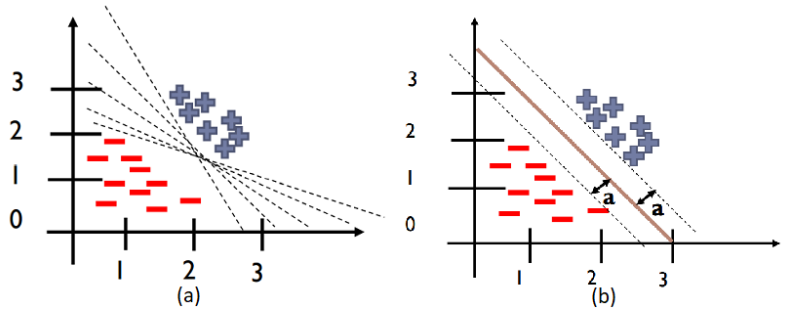

Figura 3. Máquina de Vetor de Suporte. (a) Duas classes vários separadores lineares candidatos. (b) O separador de margem máxima ao centro. Adaptado de [Nogueira 2020]

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)

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

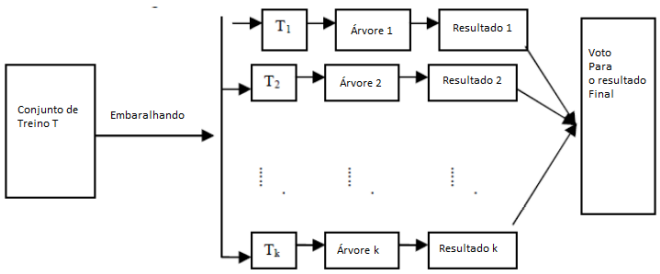

Figura 4. Modelo de Floresta Aleatória. Adaptado de [Liu et al. 2012]

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 6:

$$H(x) = arg\max_{Y} \sum_{i=1}^{k} I(h_i(x) = Y) \tag{6}$$

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 AP 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

3.1. Comparando aprendizado dos Algoritmos

4. Análise e Discussão

Section....

5. Conclusão

In tables, try to avoid the use of colored or shaded backgrounds, and avoid thick, doubled, or unnecessary framing lines. When reporting empirical data, do not use more decimal digits than warranted by their precision and reproducibility. Table caption must be placed before the table (see Table 1) and the font used must also be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Tabela 1. Variables to be considered on the evaluation of interaction techniques

	Chessboard top view	Chessboard perspective view
Selection with side movements	6.02 ± 5.22	7.01±6.84
Selection with in-depth movements	6.29±4.99	12.22±11.33
Manipulation with side movements	4.66± 4.94	3.47±2.20
Manipulation with in-depth movements	5.71 ±4.55	5.37 ±3.28

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.

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.

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.

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.

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.