Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 47 (2026), No. 7, pp. 2779–2788
DOI : 10.47974/JIOS-2332

Effectiveness of lazy algorithm-based software effort estimation

Shalu *
Department of Computer Science & Technology
Manav Rachna University
Faridabad 121004
Haryana
India

Neha Saini †
Department of Computer Science
Government College, Chhachhrauli (Yamuna Nagar)
Chhachhrauli
Yamuna Nagar 135001
Haryana
India

Aman Kumar §
Department of Computer Science Engineering
School of Engineering & Technology
Manav Rachna International Institute of Research and Studies
Faridabad 121004
Haryana
India

Ankit Gambhir ‡
Department of Computer Science & Engineering
Trinity Institute of Professional Studies
Guru Gobind Singh Indraprastha University
Greater Noida 201310
Uttar Pradesh
India

* *ORCID-ID:* 0000-0002-5516-0357

* *E-mail:* singshalu2609@gmail.com (Corresponding Author)
† *E-mail:* profnehasaini@gmail.com
§ *E-mail:* aman.kec@gmail.com
‡ *E-mail:* er.ankit.gambhir@gmail.com

© Taru Publications

Aniket Singh [@]
Department of Computer Science & Technology
Manav Rachna University
Faridabad 121004
Haryana
India

Abstract

Effort estimation in software development is crucial, since it enables a company to distribute resources efficiently, establish project schedules, and formulate budgets. Conventional approaches to the quantification of software effort occasionally rely on complex models and a detailed pre-processing of data, which makes them very resource-consuming and time-intensive. This research paper examines the feasibility of lazy algorithms as a feasible alternative method of assessing software effort in real life context. The main goal is to improve the estimating process, but maintain accuracy. The use of relaxed algorithms, including case-based reasoning and instance-based learning methods offers a distinct way of predicting software development effort. These strategies utilise real-time data with minimum pre-processing, hence providing an advantageous estimation strategy. Lazy algorithms are the ones that could quickly adjust to emerging projects and enhance the precision of forecasts as the data expands through the use of past cases and the related results. Proposed study findings indicate that it is possible to achieve the same accuracy results with lazy algorithms as with more complex models, and requires less data preprocessing and model complexity. This shows that the lazy algorithms can become a potentially successful way of estimating software effort. Out of two lazy algorithms i.e. K* and IBL empirically evaluated in this research paper, Lazy IBL has shown impressive results for three datasets i.e. NASA, Heiat Heiat and Desharnais out of four used in this research study.

Subject Classification (2020): *68N01, 68N30, 68Q25, 62P20.*

Keywords: *Software effort estimation, Lazy algorithms, Percentage of predictions (x) pred (x).*

1. Introduction

Software effort estimating (SEE) is a crucial method that facilitates the planning of software projects and assesses the time and cost necessary for a particular project [1]. The concept of SEE had been developed in the 1950s, and it has evolved to the present day to continue to fascinate software professionals who want to make accurate estimations of software labor [2], [3]. Subsequently, other models are introduced to predict software effort. Nonetheless, these models encounter numerous issues stemming from the fragmented and inconsistent nature of the information pertaining to software projects presented initially. It is exceedingly difficult to ascertain which technique is most appropriate for a specific dataset and yields the most accurate outcomes. No

[@] *E-mail:* **aniketsingh@mru.edu.in**

singular SEE method can consistently provide rapid, precise, and correct calculations in all instances [1]. The main challenge that most software companies are facing is the capability to effectively implement the program within the fixed budget and given time, and also in accordance with the demanded functionalities. Underestimation as well as overestimation is harmful to the software development. Underestimation may lead to schedule slippage and cost increase, which eventually will jeopardize quality of provided products. Nevertheless, an over estimation has the side effects of losing clients and partners [3]. Hence, precise effort estimation [4], [5] is essential for the successful implementation of a software development under the specified budget and schedule for any software enterprise [6].

The subsequent portions of the paper are organised as follows. Section 2 provides an examination of the research context. Section 3 offers a succinct overview of the research methodology. Section 4 delineates the results of the empirical assessment.

2. Literature Review

Efforts to anticipate software development efforts commonly rely on a range of methodologies that have been developed by scholars over the course of several years. The function point-based model, the constructive cost model, and the Putnam program [7] lifecycle model were among the initial models employed for estimating the effort involved in producing widely adopted software [8]. Each of these methods offered a distinct formula for determining the work required based on historical data [9]. Machine learning (ML) methods have been extensively used to predict software development labor ML algorithms enable experts to allocate more time towards focused on customers features of software systems, so allowing them to dedicate their time to the analysis of new projects. Comprehensive research was undertaken to analyze the utilization of artificial intelligence (AI) models in anticipating effort summaries for the Software Development Life Cycle. The algorithms that are frequently employed in practice include Case-Based Argumentation, Neural Networking, and Decision Tree [10].

The datasets that are discussed in a different report include Albrecht, China, Desharnais, and Kemerer. The stacking method uses generalized linear models (S-GLM) and decision trees (S-DT), support vector machines (SVM) and random forests (RF) [11]. In order to effectively estimate the software effort, author [12] Developed an RF model and enhanced its effectiveness empirically by fine tuning the critical factors. The databases were ISBSG, Tukutuku and COCOMO. An assessment was performed using the hold-out verification approach, utilising a 30 percent data subset. Three performance indicators, specifically MdMRE, MMRE, and Pred (0.25), are utilised to evaluate the efficacy of the strategy. The comparison between the suggested model and the classic regression tree shown that the improved RF model significantly outperformed the regression tree.

3. Proposed Work

Lazy Algorithms are one approach to software effort prediction introduced in this study. The lazy learners memorize the training cases without any serious exercise until the classification stage. Lazy learning training methodology is the process of delaying the adaptation process on the original data set to the time when the system is presented with a query [13]. At that stage, the system tries to conceptualize training information and then accepts queries. This is one primary benefit of using a lazy learning approach, it is capable of local approximating the desired function, similar to the k closest neighbors algorithm. The lazy learning systems can deal with multiple issues at the same time and are able to be flexible to the changes in the domain of the issue. This flexibility is obtained by the local approximations of the target function to every query requested to the system.

3.1 *IBL (Instance Based Learning)*

IBL formulates hypotheses from training instances and entails postponing processing until a new instance requires classification. The temporal complexity of this approach is contingent upon the magnitude of the training data. Whenever a new query is encountered, its previously stored information is retrieved. If there are multiple instances that meet the criteria for being the closest, the first one encountered is selected. IBL algorithms do not create concept descriptions based on specific instances. Concept descriptions are determined by the IBL algorithms' selection of similarity and classification functions, which are based on the current collection of preserved distances [14]. These functions are two out of the three constituents of the subsequent framework that delineates all IBL algorithms:

Similarity function evaluates the extent of resemblance among the training cases, represented as i, and the instances within the notion specification.

Classification function retrieves the similarity function results and the performance records of the cases in the concept description. It provides a categorization for the variable i.

Concept Description Updater: This method preserves data on classification accuracy and identifies which examples ought to be incorporated into the concept definition. The inputs consist of the categorization results, the similarity results, and a current idea description, denoted as 'i'. The function yields the altered depiction of the notion.

3.2 *K star*

The K-means algorithm is a technique for cluster analysis that entails partitioning data points into subsets based on their proximity to the group's centroid. The K* technique is defined as an IBL utilising entropy as a distance measure. K* is a straightforward learner that relies on entropy as a metric for measuring distance. One advantage is that it provides a consistent approach for managing attributes with real values, symbolic values, and absent values [15]. The categorization of new information representations, designated as x, is

determined by selecting the class with the highest occurrence among the k adjacent data elements, represented as yj where j=1, 2, 3, ..., k. Thereafter, entropic similarity is utilized to identify the most comparable cases from the collection. The application of entropic distance as a measure has numerous advantages, including the capacity to accommodate real-valued attributes and incomplete data. The K* coefficient is calculated by applying the negative natural logarithm to the likelihood P* for every transformational route from occurrence x to y, as demonstrated in equation 1.

$$K^*(yi, x) = -\ln P^*(yi, x) \quad (1)$$

The optimisation will concentrate on the % blending ratio parameter, akin to the "sphere of influence" concept in k-nearest neighbours, before evaluation with other machine learning techniques.

3.3 *Datasets Description*

Four datasets were examined for this study [16]. The assessment of ML based Lazy algorithms for effort estimation was performed utilising a well-established and esteemed dataset from a NASA software initiative. The dataset was initially presented by Bailey and Basili [17]. An aggregate of 18 observations exists, each corresponding to a distinct NASA software project. 70 percent of the dataset is designated for training, and the remaining 30% of the data is utilized for validation. The second dataset utilized is the Heiat dataset, which comprises 33 instances of software projects. The scale of a project is quantified by the number of lines of code it contains. Effort is quantified by the total hours necessary to finalise a specific project. The third dataset employed is the Miyazaki94 dataset, and the fourth is the Desharnais dataset, both of which are highly regarded in the domain of software work estimation.

3.3.1 Estimation accuracy measures

This study utilised established parameters to forecast the effort necessary for software development, specifically the mean magnitude of relative error (MMRE) and the prediction at 25 percent (Pred (0.25)). The calculation of MMRE across a repository of n items is as follows:

$$MMRE = (1/n) \sum\_(i=1)^n MREi$$

MREi denotes a standardised metric quantifying the disparity between the observed value (xi) and the expected value (xi) for observation i. The computation is executed in the subsequent manner.

$$MREi=(|xi-xi^)/xi$$

$$Pred\ (0.25) = t/q$$

Where t be the count of observations with a Mean Relative Error (MRE) <= 25 percent, and q represent the count of items of repository.

$$Pred\ (0.50) = t/q$$

Where t be the count of observations with a Mean Relative Error (MRE) <=50 percent, and q represent the count of items of repository.

$$Pred\ (0.75) = t/q$$

Where t be the count of observations with a Mean Relative Error (MRE) <=75 percent, and q represent the count of items of repository.

3.3.2 Cross validation

Cross-validation, a widely used estimation method in various fields, involves the systematic execution of repeated percentage splits. The dataset must be divided into 10 subsets, often known as "folds." Each fold is systematically reserved for testing, while the remaining nine folds are aggregated for training. This yields ten evaluation outcomes, which are subsequently averaged. In the process of "stratified" cross-validation, it is ensured during the initial division that each fold has an approximate proportion of the class values that is right. After conducting a 10-fold cross-validation and calculating the evaluation metrics, Weka proceeds to execute the learning algorithm one more (11th iteration) on the complete dataset in order to generate the model that is subsequently displayed.

4. Result and analysis

The efficacy of effort estimation for software development was assessed utilising the WEKA tool, based on several metrics including MMRE, MRE, PRED (25), PRED (50), and PRED (75). The four datasets chosen for effort prediction are NASA dataset, Heiat Heiat dataset, Miyazaki94 dataset and Desharnais dataset which are described in Table 1.

Table 1

Overview of Employed Datasets

Name of Dataset	Number of Projects	Number of Parameters
NASA Dataset	18	3 including KLOC, Methodology & Estimated Effort
Heiat Heiat Dataset	33	2 including Size and Effort
Miyazaki94	48	9 including KLOC etc.
Desharnais Dataset	81	12 including Function Points, Team Experience, language etc.

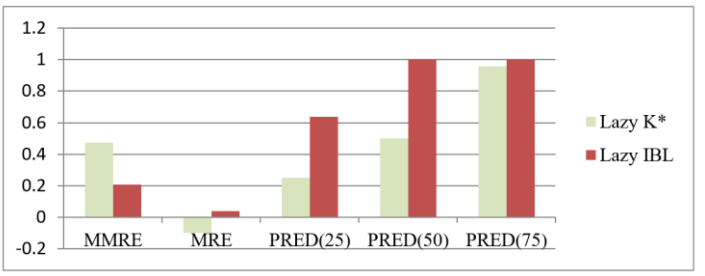

Figure 1

Analysis (NASA Dataset)

Ten cross validation method is used for cross validation. For NASA dataset, Lazy IBL demonstrates better performance compared to Lazy K* in terms of MMRE, MRE, and PRED (75) values, as shown in Figure 1. However, when considering the PRED (25) and PRED (50) values, the performance of Lazy K* is improved. Figure 2 clearly demonstrates that Lazy IBL outperforms Lazy K* on this dataset, as indicated by its MMRE value of 0.21, MRE value of 0.040, and pred (25) value of 0.636, pred (50) value of 1, and pred (75) value of 1.

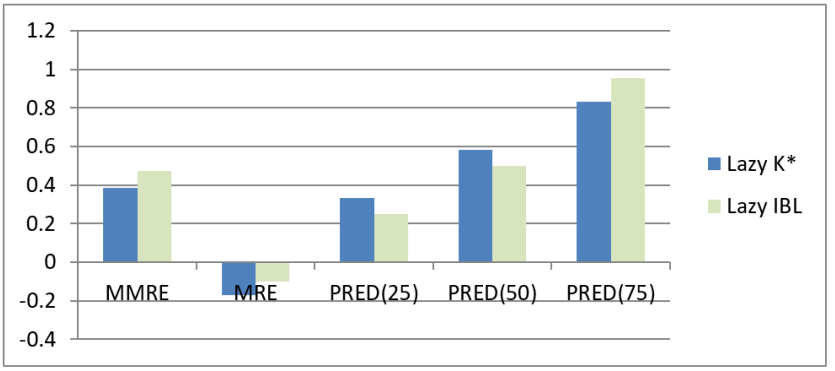

Figure 2
Analysis (Heiat Heiat Dataset)

Figure 3 illustrates that the Lazy K* method surpasses Lazy IDK regarding MMRE and MRE values for the Miyazaki94 dataset. Furthermore, Lazy K* exhibits elevated PRED (50) and PRED (75) scores.

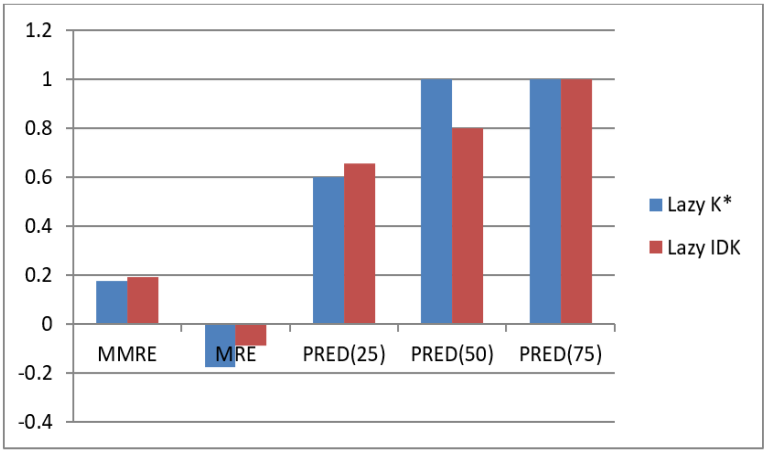

Figure 3
Analysis (Miyazaki94 Dataset)

In case of Desharnais dataset, as shown in Figure 4, Lazy IBL performs better than Lazy K*, as indicated by the MMRE and MRE values. Additionally, Lazy IBL has higher PRED values than Lazy K* algorithm.

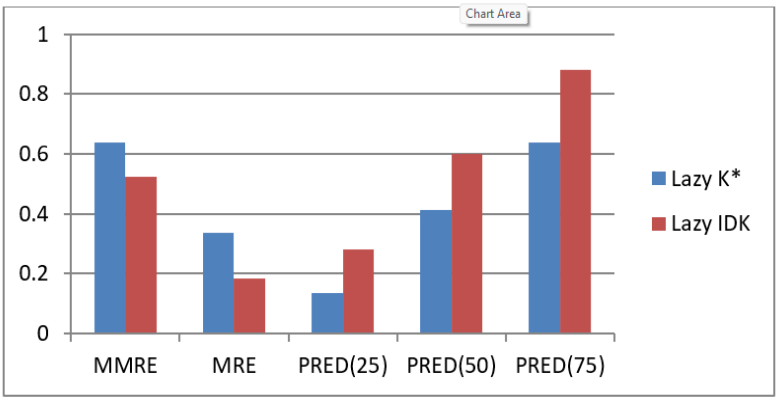

Figure 4
Analysis (Desharnais Dataset)

5. Conclusion and future work

Analysis of various datasets utilising distinct machine learning techniques reveals that the use of different algorithms to disparate datasets yields varying results. The result greatly depends on the type of data. Furthermore, it can be concluded that Lazy IBL has demonstrated commendable outcomes across three datasets: the NASA dataset, the Heiat Heiat dataset, and the Desharnais dataset utilised in our study project. Based on our study, Lazy IBL is effective in evaluating software work. This instrumentality in the use of lazy algorithms in software effort prediction is useful since it enables on-demand evaluation, which is useful in resource conservation by allowing computation to be done at the latest possible time and can also be used to estimate the effort at the latest possible time. These algorithms effectively minimize unnecessary costs, adjust to changing requirements of a project, and do not make premature promises to estimations, by postponing the delivery of correct estimates till they are needed. This is a flexible and efficient methodology and therefore most useful in dynamic and agile development environments whereby project scope and requirement can change. This enables the estimators to concentrate on the most urgent and important tasks. This thorough analysis can be done with a lot of industrial software in the future. We can also use the evolutionary methods like optimized whale algorithm, grey wolf optimization and Lion Optimization etc. on the same dataset to determine whether there are any performance improvements.

Conflicts of Interest: The authors declare no conflicts of interest.

References

[1] K. K. Aggarwal, Y. Singh, and J. K. Chhabra, "F-effort: a fuzzified model of software effort estimation," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 7, no. 3, pp. 387–400 (Jan. 2004), doi: 10.1080/09720529.2004.10698016.

[2] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data Mining Techniques for Software Effort Estimation: A Comparative Study," *IIEEE Trans. Software Eng.,* vol. 38, no. 2, pp. 375–397 (Mar. 2012), doi: 10.1109/TSE.2011.55.

[3] M. Pandey, R. Litoriya, and P. Pandey, "Validation of Existing Software Effort Estimation Techniques in Context with Mobile Software Applications," *Wireless Personal Communications,* vol. 110, pp. 1659–1677 (2020).

[4] M. Vyas and N. Hemrajani, "A novel approach for optimization of effort estimation of agile projects using SVC_RBF along with neural network backpropagation," *Journal of Information and Optimization Sciences*, vol. 43, no. 8, pp. 2089–2098 (Nov. 2022), doi: 10.1080/02522667.2022.2133216.

[5] S. Alturki and F. E-amin, "Comprehensive Analysis of Software Effort Estimation Techniques: Evolving Trends, Key Challenges, and Prospective Directions," *International Journal of Computer Applications,* vol. 186, no. 68, pp. 42–48 (2025).

[6] P. V. Terlapu, K. K. Raju, G. Kiran Kumar, G. Jagadeeswara Rao, K. Kavitha, and S. Samreen, "Improved Software Effort Estimation Through Machine Learning: Challenges, Applications, and Feature Importance Analysis," *IEEE Access,* vol. 12, pp. 138663–138701 (2024), doi: 10.1109/ACCESS.2024.3457771.

[7] C. Singh, K. Kamini, N. Juneja, P. K. Joshi, and R. Garg, "Performance comparison of Putnam model using new technology trends for software maintenance cost estimation," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 25, no. 3, pp. 691–703 (Apr. 2022), doi: 10.1080/09720529.2021.2016220.

[8] R. P. S. Bedi and A. Singh, "A novel technique for software effort estimation based on artificial neural network," *International Journal of Applied Engineering Research*, vol. 12, no. 1, pp. 144–151 (2017).

[9] A. Najm, A. Zakrani, and A. Marzak, "Systematic Review Study of Decision Trees based Software Development Effort Estimation,"

International Journal of Advanced Computer Science and Applications, vol. 11, no. 7, Art. no. 7 (2020), doi: 10.14569/IJACSA.2020.0110767.

[10] S. S. Gautam and V. Singh, "Adaptive discretization using golden section to aid outlier detection for software development effort estimation," *IEEE Access,* vol. 10, pp. 94839–94860 (2022), doi: 10.1109/ACCESS.2022.3200149.

[11] P. V. Arun Gopal, A. K. Kanchana, and V. Varadarajan, "Estimating software development efforts using a random forest-based stacked ensemble approach," *Electronics*, vol. 10, no. 10, Art. no. 1195 (2021), doi: 10.3390/electronics10101195.

[12] Z. abdelali, H. Mustapha, and N. Abdelwahed, "Investigating the use of random forest in software effort estimation," *Procedia Computer Science,* vol. 148, pp. 343–352 (Jan. 2019), doi: 10.1016/j.procs.2019.01.042.

[13] S. Vijayarani and M. Muthulakshmi, "Comparative Analysis of Bayes and Lazy Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 8 (2013).

[14] I. M. Galván, J. M. Valls, N. Lecomte, and P. Isasi, "A Lazy Approach for Machine Learning Algorithms," in *Artificial Intelligence Applications and Innovations III,* I. G. Maglogiannis, V. Koutkias, K. Vlahavas, and I. Ch. Vlahavas, Eds., IFIP Adv. Inf. Commun. Technol., vol. 296, Boston, MA: Springer, pp. 517–522 (2009), doi: 10.1007/978-1-4419-0221-4_60 .

[15] P. Tamrakar and SP S. Ibrahim, "Comparative Study of different Lazy Learning Associative Classification Methods," *Procedia Computer Science,* vol. 165, pp. 370–376 (2019).

[16] M. F. Bosu and S. G. Macdonell, "Experience: Quality Benchmarking of Datasets Used in Software Effort Estimation," *J. Data and Information Quality*, vol. 11, no. 4, pp. 1–38 (Dec. 2019), doi: 10.1145/3328746.

[17] J. W. Bailey and V. R. Basili, "A meta-model for software development resource expenditures," in *Proc. 5th Int. Conf. Softw. Eng. (ICSE '81)*, San Diego, CA, USA: IEEE Press, pp. 107–116 (1981), doi: 10.5555/800078.802522.

Received March, 2026