International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Study of Evaluation Measures for Software Effort Estimation Using Machine Learning

[1]Rajani Kumari Gora [2]Prof. Ripu Ranjan Sinha

Submitted: 11/02/2023 **Revised:** 16/04/2023 **Accepted:** 07/05/2023

Abstract- Software effort estimation is a crucial process which involves predicting how much time and money will be needed to accomplish a software development project. Expert opinion and past data are used in conventional estimating techniques, which may be inefficient and prone to mistakes. Machine learning offers a promising approach to automate this process by learning from past projects and predicting effort estimates for new projects. Using machine learning, this article takes an in-depth look at the practise of software work estimation. In this study, several machine learning models, including support vector machine, KNN, ANN, linear regression, support vector machine, and neural network, are trained and evaluated on a dataset of software projects. This paper also presents some comparative results of the various machine learning algorithms, including multilayer perceptron (MLP), support vector machine (SVM), and linear regression (LR), showing that the MLP model achieves the lowest MMRE value, 13%, while the SVM achieves the highest PRED(25), 87.65%, for software effort estimation.

Keywords— *Software development, Effort estimation, Software development life cycle (SDLC), SDLC models, Machine learning*

1. Introduction

A software system's development time may be estimated by software development effort estimation (SDEE). When broken down into its component parts, software development, and maintenance effort estimate (SDEE) may be thought of as a sub-domain of software engineering. The words software cost estimate and software effort estimates are sometimes used interchangeably in the academic community, despite the fact that the bulk of software expenses are attributed to effort. Project success relies heavily on accurate initial development effort estimates made early in the software life cycle [1]. Estimating materials and time required to finish a software project in accordance with the planned schedule and budget is known as software effort estimation. Man-hours and man-months are common measures of labour. Estimating how much time and energy will go into creating a piece of software is a common first step in the software development process. Research into software effort estimation has been ongoing since the 1960s due to the multiple challenges inherent in producing reliable estimates. Although 69percent of finished software projects achieved their intended business goals, 43percent went over budget, 48percent were delivered late, and 15 percent were rated a failure due to faulty effort estimates, according to study conducted by Project Management Institute (PMI) in 2017[2]. [3] three types of learning: algorithmic, non-algorithmic, and machine learning. Estimating software development time using an algorithmic approach relies on a statistical and mathematical framework. Estimating methods include COCOMO-II (Constructive Cost Model), FPA (Function Point Analysis), UCP (Use Case Point), Putnam Software Life cycle Management (SLIM), and COCOMO-I. Analytical evaluations and interpretations provide the basis of nonalgorithmic models. Estimating how much work a project will take is crucial to the success of any software development endeavor. The term "effort estimate" refers to the approach used to determine amount of time and other resources requirements for achieving a goal and delivering a product or service which fulfills customer's specified requirements. To aid in project planning and ensure a smooth rollout, software engineers need accurate effort estimate models[4] Software development projects may be more prosperous if effort estimates are spot-on, but if they're off, the company might lose money on its marketing and sales.

2. Software Effort Estimation

Software project management relies heavily on accurate estimates of time and resources spent developing software. (SPM). It is also crucial that prediction methods can be trusted to provide accurate results. Accurate effort appraisals, especially at the beginning of a project, may help reduce the high risks associated with developing a software product. The widespread use of unreliable estimate methods causes most projects to go over budget and over schedule. The use of ML (machine learning) in

1Computer Science
S. S. Jain Subodh P.G. College,
Rajasthan Technical University, Kota
rajanigora@gmail.com
2Computer Science
S. S. Jain Subodh P.G. College,
Rajasthan Technical University, Kota
drsinhacs@gmail.com

software project estimating, however, may result in more accurate work estimation.[5].

Estimating a software project requires calculating how long it will take, how many people will be needed, how much it will cost, and when it will be finished[6].

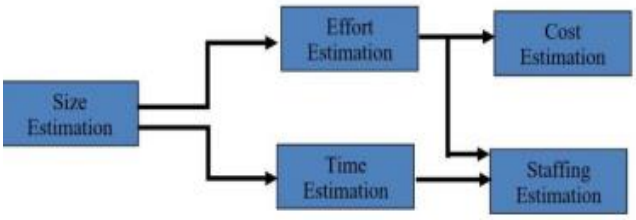

Fig. 1. Type of software estimation process [7]

1. Estimating size

The first step in creating a meaningful project estimate is to create a precise software size estimate. Scope and size estimation can begin in tandem with the formal specifications of the project criteria.

2. Estimating effort

Once you have a ballpark idea of how big a product is going to be, putting together a time estimate is a breeze. It is only feasible to convert software size to total project work estimate if the software development lifecycle of a project has been determined.

3. Estimating schedule

The third step is to use the effort estimate to create a schedule for the software development project. It's important to think about how many people are working on a project, what kind of work they're doing, how long it's going to take, and when it's supposed to be finished.

4. Staffing Estimation

When it comes to software project planning, Staffing Estimation is all about using the right model to solve the human resource allocation issue[8]

5. Estimating Cost

The whole cost of a project may be broken down into its component parts, which might include things like labour, hardware, software, or leasing fees, meetings, testing, communications, training, office space, etc. All of the aforementioned elements have some bearing on the evaluation of effort.[9]

A. Techniques for estimating software work fall into 3 primary categories:

1. Expert Judgment

The most common approach to software cost assessment is the expert judgement methodology, in which actual experts are tasked with making estimates about the product's size and price. The methodology was developed from the project manager's prior work on other software projects like this one. Expert cost-estimating methods are useful when time and resources are limited [10]. Professional opinion might be flawed due to human fallibility and prejudice. The effectiveness of the method relies on the opinion of experts, who may have different levels of expertise with the same sort of project, leading to different cost estimates. It is most useful, however, in smaller to medium-sized software projects when there has been little to no change in the development teams or software characteristics from earlier projects.

2. Algorithmic Estimation

The algorithmic method gives a set of mathematical formulas that may be used for software estimation. These mathematical estimates are based on research and historical data, and they include variables such as number of functions, source lines of code, and other cost factors including language, design technique, etc [11].

3. Machine Learning

The majority of software cost assessment methods rely on statistical approaches, which lack the rigour and logic to provide reliable outcomes. Due to its potential to improve estimate accuracy via the training of estimating rules and the repetition of run cycles, ML technologies may be well-suited to this domain. Popular ML methodologies include Rule Induction, Genetic Algorithm, ANN, case-based reasoning, trees for regression and classification, SVM (support vector machines), with augmenting, and MART[12]. It's not easy to tell which method produces the most reliable results for a given dataset. Nonetheless, a lot of work has been done on estimate approaches using ML, and the literature shows that ML methods may provide appropriate estimation models.

3. Software Development Life Cycle

Systems Development Life Cycle (SDLC), sometimes known as Software Development Life Cycle, is a phrase used in domains of systems engineering, data management, and software engineering to describe phases involved in the design and implementation of novel or improved systems. The term is most often used in reference to digital infrastructures. There are several SDLC-based software development approaches that are fundamental to the field of software engineering. Methodologies such as these provide a structure for organizing and directing development of an information system's software[13].

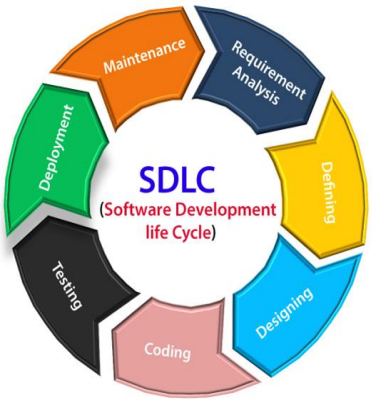

Fig. 2: Software Development Life Cycle

The many SDLC stages are shown on cyclical diagram above. The steps are as follows:

1. Requirement Analysis
2. Defining
3. Designing
4. Coding
5. Testing
6. Development
7. Maintenance.

A. SDLC MODELS

There are different models explained in detail below:

a) Waterfall Model

This is a sequential model, in which procedures are completed in order listed. There is a pattern here. Stage one's outputs "flow" into stage two's inputs, which "flow" into stage three's inputs, and so on. In addition, users can't go on to the next level until current one is finished. [14].

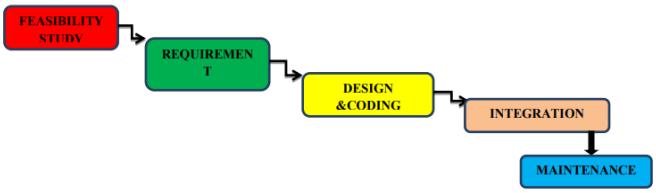

Fig. 3: Waterfall Model

b) Spiral Model

It's similar to incremental approach, but it puts more emphasis on analyzing risks. Planning, Risk assessment, Engineering, Construction, and finally, Release are the four components. These stages are repeated indefinitely in the program since they correlate to the many model spirals. In first stage, called "planning," requirements are gathered In the risk phase, problems are analyzed and potential remedies are proposed. The final product of engineering is software that has been thoroughly tested. Finally, the client assesses the results of the program.

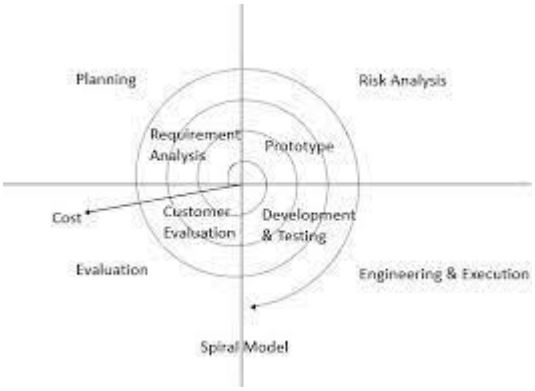

Fig. 4: Spiral Model

c) V Shaped Model

V-shaped model develops further from waterfall approach. The V-shaped model illustrates the interdependencies between various stages of development and corresponding testing stages. The verification and validation model is another name for this framework. That's because there's a validation step for every verification step. Checking the software's efficiency is the most important part of using it. It has to be tried and tried again. As a result, the V-shaped approach places primary emphasis on testing. Although verification checks for bugs in the code, validation makes sure it was written properly. A lot of iteration and testing goes into this procedure. The 'tester's life cycle' outlines these steps. As it is still difficult to make adjustments after the fact, this approach is best used when there are no unanticipated requirements. The model devotes majority of its attention to testing phase, which corresponds to each phase[15].

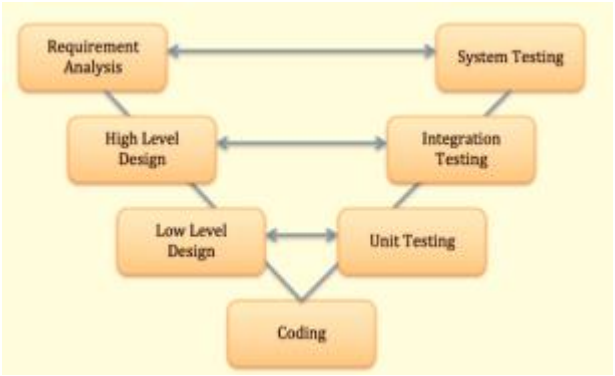

Fig. 5: V-Shaped Model

d) Iterative SDLC Model

The Iterative SDLC model does not require a comprehensive list of requirements prior to project commencement. The development process may begin with functional component's needs, that can be extended subsequently. The procedure is repeated, allowing for creation of new product variants with each cycle. Every iteration (which lasts from 2 to 6 weeks) comprises the creation of a distinct system component, which is then added to previously produced functionality. Mathematically speaking, iterative model is a realization

of sequential approximation technique, which results in a progressive approach to the desired end product form. [16].

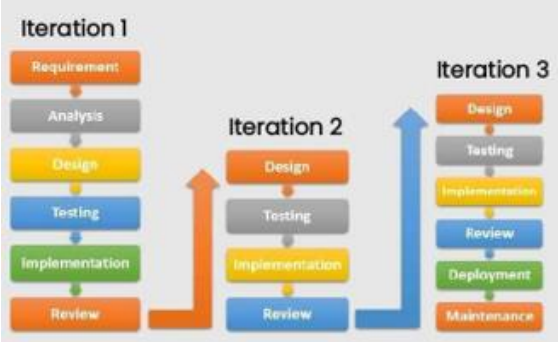

Fig. 6: Iterative Model

4. Machine Learning-Based Software Effort Estimation

Rather of being explicitly programmed by a human expert, algorithms in the area of ML are trained to carry out tasks by inferring the appropriate steps based on the data they are exposed to. The most popular approach is supervised, although semi-supervised and unsupervised methods also exist (20). In supervised learning, an algorithm is trained on a labelled dataset before being asked to classify an unlabeled data point. For instance, software may be taught to recognize cancerous from noncancerous skin lesions by exposing it to a series of images of lesions that have been manually identified as such. After training the algorithm with these pictures, it would be evaluated by presenting it with new, unlabeled photos and asking it to categorize them as benign or malignant. [17] Unsupervised learning is the process of learning from data without the use of a training dataset. The goal of both supervised and unsupervised learning is to label incoming data with appropriate categories. In contrast to supervised learning, unsupervised learning relies on a model's ability to classify data solely on the basis of its intrinsic properties rather than labels applied to the data at input. With this ML technique, we are able to take a more exploratory tack in order to discover hidden patterns in the data's distribution. Finally, semi supervised ML combines aspects of supervised and unsupervised learning into a single methodology. To simplify data labelling, this method combines a high number of unlabeled inputs with a small number of labelled inputs.[18].

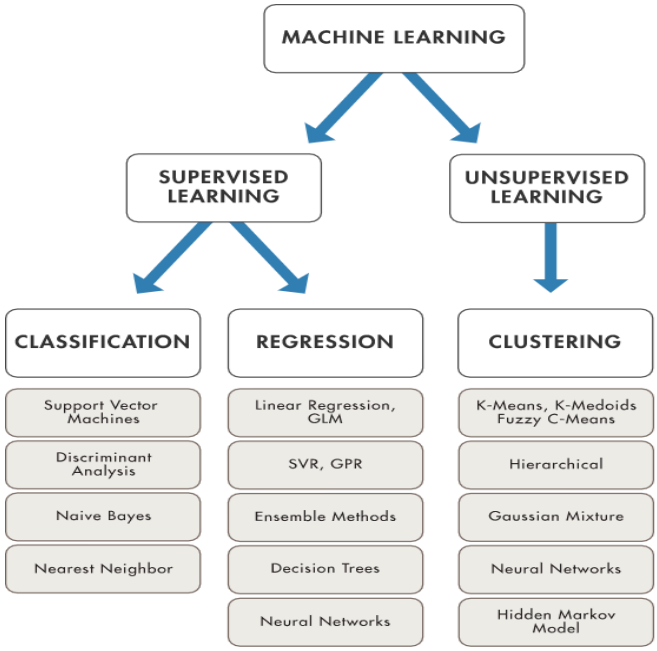

Fig. 7: Machine Learning

- **K-Nearest Neighbor (KNN)**

k-NN is a simple nonparametric method that does not need any familiarity with the data distribution in advance. The method relies on the Euclidean distance principle, which states that data points with similar characteristics tend to cluster together in a dataset. If the instances have been labelled with categories, then the label of an unlabeled instance can be inferred from the labels of its neighbours. The modelling procedure is carried out three times (repeats=3), with k set to 10 (number=10) each time, and average of results is used[19].

- **Linear Regression**

When dependent variable is anticipated to be a linear mixture of another properties, linear model of regression known as linear regression may be used. The purpose of linear regression is to minimise discrepancy between the observed and predicted values by adjusting the coefficients in a linear model.

Some of the benefits of LR are:

- The LR statistical formulas are straightforward and simple.
- LR makes it simple to interpret data.

Linear regression is a straightforward method with few restrictions. Yet, the goal feature and its dependent characteristics were chosen with great care. [20].

- **MLP (Multi-Layer perceptron)**

Multi-Layer perceptron is yet another supervised learning technique that relies on a function f(x): Using the formula, one may learn from a training dataset RmR0where m represents the quantity of input dimensions and o represents the quantity of output dimensions. In order to

solve a regression or classification problem, the MLP may be taught a non-linear function by providing it with a set of features, X= x1, x2,..., xm, and an objective, y. MLPs diverge from other methods like Linear Regression in large part because of hidden layers, which are non-linear and located between input & output layer [21]. This is a one-hidden-layer Multi-Layer Perceptron (MLP) example:

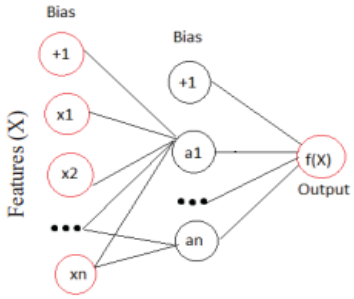

Fig. 8: Multilayer perceptron

- **Artificial Neural Networks (ANN)**

ANN are mathematical or computer models inspired by the human brain. ANN can be trained to perform a variety of tasks, including pattern recognition and data classification. ANNs consist of both neurons (the nodes) and synapses (the connections between them), much like biological neural networks (weights). For each neuron, there are n (the number of neurons in layer below) synapses (inputs) with n [22].

- **Support Vector Machine (SVM)**

SVM is a prominent ML approach for regression and classification applications [23]. In infinite or high-dimensional space, a support vector machine (SVM) creates a hyper-plane or set of hyper-planes. When an improvement in margin leads to a reduction in generalization error of a classifier, hyper-plane which separates classes by largest average distance from their nearest training data points is winner. Effective in high-dimensional spaces, its behaviour depends on the mathematical function used to define the kernel. You may use a radial basis function (RBF), sigmoidal, linear, polynomial, etc.

- **Naive Bayes (NB)**

NB algorithm takes its cues from Bayes' theorem and operates under the premise of feature independence. It performs extremely well for both multi-class and binary classifications in many real-world applications, such as document or text categorization, spam filtering, etc. The NB classifier is a powerful tool for data classification and creating trustworthy prediction models.[24]. To quickly and reliably estimate the necessary parameters, unlike more sophisticated algorithms, just a small quantity of training data is needed. Yet, its robust assumptions on the independence of characteristics may compromise its performance. Some frequent forms of NB classifiers include Categorical, Multinomial, Complement, Gaussian, and Bernoulli.

5. Literature Review

In this article, they'll take a look back at some of the latest studies into the art of software project effort estimate. Many new studies have been published in this field of study. In this article, they look at many articles that have utilized ML methods to predict how long a software project would take to complete.

Ritu and Garg, (2022) Naive Bayes, Random Forests, Logistic Regression, stochastic gradient boosting, decision trees, and narrative points for estimate are only few of the ML methods suggested. Increasingly, businesses in the software industry are turning to non-parametric and non-traditional methods for estimating software development projects in order to make up for their shortcomings. In this research, they provide and analyse a comparison of the aforementioned methods in order to provide an in-depth analysis of their relative merits [25].

Assefa et al (2022).'s primary motivation for computing three distinct ML algorithms is to identify the one that yields the most accurate predictions of future effort. They have used SVM, multi-layer perception and linear regression, algorithms on SEERA (Software Engineering in the Republic of Sudan) data set to predict the time and effort needed to complete the project. To further assess the precision with which the predictive model anticipates the time and resources required to create software, they have explored the assessment metrics of MSE (mean square error), MAE (Mean absolute error), and R-squared. Experiments were conducted on all of the chosen ML algorithms in a Jupyter notebook. Therefore, the linear regression R-square score is 0.95, the multi-linear regression score is 0.83, and the SVR score is less than 0.04. The findings contrast the predictive abilities of the MLP and SVR models and demonstrate that the linear regression model is superior[26].

There are many different ML methods that may be utilized to estimate work, and Brar and Nandal (2022) analyse some of them. There has been a rise in research over the last two decades into using ML methods to improve the precision of effort estimations. Predicting labour needs may be done using a variety of estimated methods including COCOMO, analogy-based estimate, expert judgement, Putnam model, and ML. Substantial software project risks came from the algorithmic models' poor accuracy and unstable design. So, it is crucial to make yearly cost estimates for the project and evaluate them against other alternatives. Nevertheless, ML has its limitations in terms of effort prediction since no one approach can be considered optimal. The primary goal of this study is to provide an overview of the current state of

the art of several ML methods for assessing levels of effort. [27].

Scrum-based Agile projects that are created over the course of numerous sprints are the focus of this article by Govil and Sharma (2022). They proposed a tweak to the existing method that would account for an extra 36 success factors and offer a cost and time estimate for bringing the project to fruition. For this study and calculation, they utilised a dataset of 30 projects, each of which was classified as either "low," "mid," or "high". Experts verify the accuracy of this data collection. They also compared our findings to the current method and discovered that, while taking into account more success variables, they are more cost-effective and take less work[28].

Based on their findings, Jang and Wu (2022) suggest that hardware development might benefit from using software effort estimate. They utilise ML and deep learning to precisely control product development time by estimating how long it will take to complete various jobs throughout the hardware development process. In this study, natural language processing (NLP) is used to extract the keywords of the development process challenges using semantic analysis, which are then used as features in subsequent analysis. Several ML models, including random forest, decision tree, XGBoost, and RNN, are used to predict a time period and their accuracy, MMRE, and PRED are compared (25). The decision tree outperformed the other three models in the experiments. This research demonstrates that task tracking during hardware development may benefit from the software effort estimate method[29].

Setiadi et al. (2021) used a publicly available data set to conduct an attribute selection experiment with Particle Swarm Optimization (PSO) for the project's parameters based on an estimation made with the K-NN algorithm. Kitchenham CSC Desharnais, Maxwell, and Kamrer were used as software estimation effort datasets for this research. According to the study's findings, the RMSE value may be decreased using PSO (Particle Swarm optimization's) feature selection. This demonstrates which RMSE's precision increases as its value decreases. This study is helpful because it can be used by programmers to better anticipate how long it will take to complete an application so that it will function as intended[30]

6. Results Illustration

A. Performance Matrix

In this section, we will present results for the experiments we have done using four ML techniques such as MLP (multilayer perceptron), SVM, linear regression, and ensemble learning.

A. MMRE

Mean Magnitude of Relative Error is abbreviated as MMRE. It's a common way to measure how well software effort estimation models do their job. The MMRE calculates an estimate's average percentage error, which is the absolute difference between estimated and actual effort dividing by actual effort, on the whole.

It is described as:

$$MMRE = (1/n) * \Sigma(|Ei|/Ai)$$

where n is the number of estimates, Ei is the difference between the estimated effort and the actual effort for the i-th estimate, and Ai is the actual effort for the i-th estimate. The absolute value of the difference is taken to ensure that errors in both directions (overestimation and underestimation) are treated equally. For accurate predictions, a low MMRE score is ideal, and vice versa[31].

n = Total number of data points.

Ai =Actual value of the data point.

B. PRED 25

PRED (25) is a performance assessment statistic that assesses the proportion of projected values within 25 percent of actual value. It is often used in the context of predicting or predictive modeling, where the aim is to properly anticipate future values based on previous data.

Once the threshold is defined, we can calculate PRED (25) as follows:

PRED (25) = (number of correct predictions / total number of predictions) * 100%

Table 1:Comparision of different ML algorithms

Algorithm	MMRE	PRED (25)
Multilayer perceptron[32]	15.1	87.65
SVM[33]	13	76.91
Linear regression[34]	15.0	71.42

The results of Table 1 show that we can compare the performance of four different algorithms on a predictive modeling task using two evaluation metrics: Measures of accuracy include MMRE and PRED (25) percentage of predicted values that are within 25% of actual value.

MLP got 15.1% MMRE and 87.65% PRED (25), whereas SVM achieved MMRE 13% and 76.91% PRED (25). Linear regression got 15.0% MMRE and 71.42 PRED (25). As can be seen from the findings, SVM achieved the lowest MMRE(13%) and greatest PRED(25)(76.91%) compared to the other algorithms. MLP also performed well with a high PRED(25) of 87.65%, but had a higher

MMRE than SVM. Linear regression performed better than MLP in terms of PRED(25) but had a similar MMRE.

Overall, the choice of algorithm depends on specific tasks and trade-off between accuracy and computational efficiency. In this case, SVM would be the recommended algorithm due to its overall better performance on both evaluation metrics.

MMRE and 71.42 PRED (25).

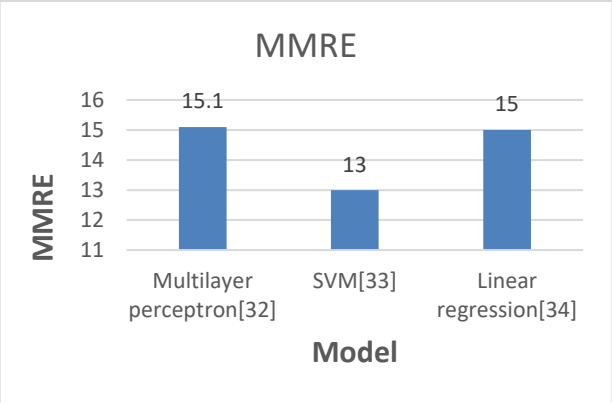

Fig. 9: MMRE values for the proposed model

Fig 9 shows the comparison of MMRE for MLP, SVM, linear regression, and ensemble learning classifier models. MLP 15.1 has a higher MMRE than the other model.

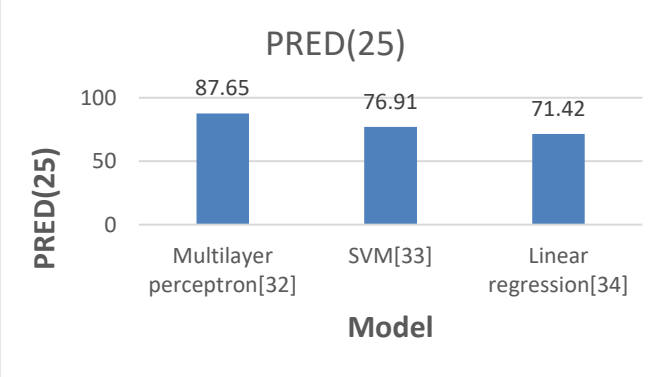

Fig. 10: PRED (25) values for the proposed model

Fig 10 shows the comparison of PRED (25) for MLP, SVM, linear regression, and ensemble learning classifier models. MLP 87.65 has a higher PRED (25) than the other model.

7. Conclusion

Estimating how much time and money will be needed to complete a software project is called an "effort estimate," and it is an essential part of the software development process. The purpose of effort estimation is to provide reliable forecasts of time, money, and other assets needed to accomplish a software development project. Region of uncertainty- Uncertainty at the outset of the endeavor is the greatest challenge. Often, not even the customer has a thorough understanding of the requirements. This paper provides a detailed discussion about the software effort estimation field, and SDLC models, and also this paper compares various ML techniques for software effort estimation prediction. A low MMRE value indicates that the estimates are generally accurate, while a high MMRE value indicates that the estimates are generally inaccurate. A high PRED (25) value indicates that a high percentage of estimates fall within 25% of the actual effort, which suggests that the estimates are sufficiently accurate for practical purposes. The use all machine learning algorithms provides efficient results but the MLP, and SVM model outperforms the other model with lowest 13% of MMRE, and highest 87.65% of PRED (25).

References

[1] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology*. 2012, doi: 10.1016/j.infsof.2011.09.002.

[2] S. M. Satapathy, B. P. Acharya, and S. K. Rath, "Early stage software effort estimation using random forest technique based on use case points," *IET Softw.*, 2016, doi: 10.1049/iet-sen.2014.0122.

[3] Z. Dan, "Improving the accuracy in software effort estimation: Using artificial neural network model based on particle swarm optimization," 2013, doi: 10.1109/SOLI.2013.6611406.

[4] Tung Khuat and Hanh Le, " An Effort Estimation Approach for Agile Software Development using Fireworks Algorithm Optimized Neural Network.," *Int. J. Comput. Sci. Inf. Secur.*, 2016.

[5] M. N. Mahdi *et al.*, "Software Project Management Using Machine Learning Technique—A Review," *Appl. Sci.*, vol. 11, no. 11, 2021, doi: 10.3390/app11115183.

[6] A. T. Raslan and N. R. Darwish, "An enhanced framework for effort estimation of agile projects," *Int. J. Intell. Eng. Syst.*, 2018, doi: 10.22266/IJIES2018.0630.22.

[7] A. A. Abdulmajeed, M. A. Al-Jawaherry, and T. M. Tawfeeq, "Predict the required cost to develop Software Engineering projects by Using Machine Learning," 2021, doi: 10.1088/1742-6596/1897/1/012029.

[8] N. A. Mohamed, A. Al-Qasmi, S. Al-Lamki, M. Bayoumi, and A. Al-Hinai, "An estimation of staffing requirements in primary care in Oman using the workload indicators of staffing needs method," *East. Mediterr. Heal. J.*, 2018, doi: 10.26719/2018.24.9.823.

[9] E. Meenakshi and E. Sumeet, "Review on Software

Effort estimation by machine learning Approaches," vol. 9028, pp. 2002–2005, 2018.

[10] J. G. Borade and V. R. Khalkar, "Software Project Effort and Cost Estimation Techniques," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, 2013.

[11] Rekha T. and R. P.K., "Machine Learning Methods of Effort Estimation and It's Performance Evaluation Criteria," *Int. J. Comput. Sci. Mob. Comput.*, 2017.

[12] R. Malhotra and A. Jain, "Software Effort Prediction using Statistical and Machine Learning Methods," *Int. J. Adv. Comput. Sci. Appl.*, 2011, doi: 10.14569/ijacsa.2011.020122.

[13] P. Seema Suresh Kute and P. Surabhi Deependra Thorat, "A Review on Various Software Development Life Cycle (SDLC) Models," *Int. J. Res. Comput. Commun. Technol.*, 2017.

[14] A. M. Kale, V. V Bandal, and K. Chaudhari, "A Review Paper on Software Testing," *Int. Res. J. Eng. Technol.*, vol. 5, no. 2, p. 1268, 2008, [Online]. Available: www.irjet.net.

[15] G. Gurung, R. Shah, and D. P. Jaiswal, "Software Development Life Cycle Models-A Comparative Study," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, no. July 2020, pp. 30–37, 2020, doi: 10.32628/cseit206410.

[16] B. Tarika, "A Review of Software Development Life Cycle Models," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, pp. 62–68, 2019, doi: 10.31058/j.data.2019.34002.

[17] M. M. Moore, E. Slonimsky, A. D. Long, R. W. Sze, and R. S. Iyer, "Machine learning concepts, concerns and opportunities for a pediatric radiologist," *Pediatr. Radiol.*, 2019, doi: 10.1007/s00247-018-4277-7.

[18] G. Chartrand *et al.*, "Deep learning: A primer for radiologists," *Radiographics*. 2017, doi: 10.1148/rg.2017170077.

[19] A. Akella and S. Akella, "Machine learning algorithms for predicting coronary artery disease: Efforts toward an open source solution," *Futur. Sci. OA*, 2021, doi: 10.2144/fsoa-2020-0206.

[20] A. Kanneganti, "Using Ensemble Machine Learning Methods in Estimating Software Development Effort," no. October, 2020.

[21] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, 2011.

[22] M. Humayun and C. Gang, "Estimating Effort in Global Software Development Projects Using Machine Learning Techniques," *Int. J. Inf. Educ. Technol.*, 2012, doi: 10.7763/ijiet.2012.v2.111.

[23] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Comput. Sci.*, 2021, doi: 10.1007/s42979-021-00592-x.

[24] I. H. Sarker, "A machine learning based robust prediction model for real-life mobile phone data," *Internet of Things (Netherlands)*, 2019, doi: 10.1016/j.iot.2019.01.007.

[25] Ritu and Y. Garg, "Comparative Analysis of Machine Learning Techniques in Effort Estimation," in *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, 2022, vol. 1, pp. 401–405, doi: 10.1109/COM-IT-CON54601.2022.9850592.

[26] Y. Assefa, F. Berhanu, A. Tilahun, and E. Alemneh, "Software Effort Estimation using Machine learning Algorithm," in *2022 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, 2022, pp. 163–168, doi: 10.1109/ICT4DA56482.2022.9971209.

[27] P. Brar and D. Nandal, "A Systematic Literature Review of Machine Learning Techniques for Software Effort Estimation Models," in *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, 2022, pp. 494–499, doi: 10.1109/CCiCT56684.2022.00093.

[28] N. Govil and A. Sharma, "Estimation of cost and development effort in Scrum-based software projects considering dimensional success factors," *Adv. Eng. Softw.*, vol. 172, p. 103209, 2022, doi: https://doi.org/10.1016/j.advengsoft.2022.103209.

[29] H.-C. Jang and S.-C. Wu, "Tracking of Hardware Development Schedule based on Software Effort Estimation," in *2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2022, pp. 305–310, doi: 10.1109/IEMCON56893.2022.9946524.

[30] A. Setiadi, W. F. Hidayat, A. Sinnun, A. Setiawan, M. Faisal, and D. P. Alamsyah, "Analyze the Datasets of Software Effort Estimation With Particle Swarm Optimization," in *2021 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 2021, pp. 197–201, doi: 10.1109/ISITIA52817.2021.9502208.

[31] and Z. J. Israr ur Rehman, Zulfiqar Ali, "An

Empirical Analysis on Software Development Efforts Estimation in Machine Learning Perspective," *ADCAIJ Adv. Distrib. Comput. Artif. Intell.*, vol. 10, 2021.

[32] N. Saini and B. Khalid, "Empirical Evaluation of machine learning techniques for software effort estimation," *IOSR J. Comput. Eng.*, 2014, doi: 10.9790/0661-16193438.

[33] H. M. Premalatha and C. V. Srikrishna, "Effort estimation in agile software development using evolutionary cost- sensitive deep Belief Network," *Int. J. Intell. Eng. Syst.*, vol. 12, no. 2, pp. 261–269, 2019, doi: 10.22266/IJIES2019.0430.25.

[34] M. Vyas and N. Hemrajani, "Predicting effort of agile software projects using linear regression, ridge regression and logistic regression," *Int. J. Tech. Phys. Probl. Eng.*, 2021.

[35] B. Marapelli*, "Software Development Effort Duration and Cost Estimation using Linear Regression and K-Nearest Neighbors Machine Learning Algorithms," *Int. J. Innov. Technol. Explor. Eng.*, 2019, doi: 10.35940/ijitee.k2306.129219.

[36] Alaria, S. K. "A.. Raj, V. Sharma, and V. Kumar."Simulation and Analysis of Hand Gesture Recognition for Indian Sign Language Using CNN"." *International Journal on Recent and Innovation Trends in Computing and Communication* 10, no. 4 (2022): 10-14.

[37] Satish Kumar Alaria. Design & Analysis of Cost Estimation for New Mobile-COCOMO Tool for Mobile Application. *IJRITCC* 2019, *7*, 27-34.

[38] Najneen Qureshi, Manish Kumar Mukhija and Satish Kumar, "RAFI: Parallel Dynamic Test-suite Reduction for Software", New Frontiers in Communication and Intelligent Systems, SCRS, India, 2021, pp. 165-176. https://doi.org/10.52458/978-81-95502-00-4-20.