An Investigation into the Suitability of k-Nearest Neighbour (k-NN) for Software Effort Estimation

Razak Olu-Ajayi
Department of Computer Science,
University of Hertfordshire
Hatfield, UK

***Abstract*—Software effort estimation is an increasingly significant field, due to the overwhelming role of software in today's global market. Effort estimation involves forecasting the effort in person-months or hours required for developing a software. It is vital to ideal planning and paramount for controlling the software development process. However, there is presently no optimal method to accurately estimate the effort required to develop a software system. Inaccurate estimation leads to poor use of resources and perhaps failure of the software project. Effort estimation also plays a key role in deducing cost of a software project. Software cost estimation includes the generation of the effort estimates and project duration to predict cost required to develop software project. Thus, effort is very essential and there is always need to enhance the accuracy as much as possible. This study evaluates and compares the potential of Constructive COst MOdel II (COCOMO II) and k-Nearest Neighbor (k-NN) on software project dataset. By the analysis of results received from each method, it may be concluded that the proposed method k-NN yields better performance over the other technique utilized in this study.**

Keywords*—Software effort estimation; machine learning; k-Nearest Neighbor; Constructive COst MOdel II

I. INTRODUCTION

Since the invention of computers, a vast number of people find themselves reliant on computers. Computers are appearing in nearly every aspect of our lives, such as transportation, banking, education, communication as well as personal health. In general, computers are making things easier for us, for example, working electronically from home, socialising with long distance friends. While a computer is merely a box of circuits that achieve software level tasks for its user, software is simply a set of instructions which enables the computer to perform specified tasks.

Despite the growing popularity of software, there are still issues been encountered in various aspect of its development which has been receiving attention from several researchers. In 1968, software engineering emerged at a meeting in a discussion of what was then called 'software crisis' [1]. It became apparent that developer approaches to software development did not scale up to large and complex software systems. These issues were unreliable, cost overrun, and late delivery [2]. Many software projects still suffer from inaccurate estimation hence they are delivered late or worse still abandoned.

For example, in April 1990, the Regional Information Systems Plan (RISP) for the Wessex Regional Health Authority was abandoned, five years after it started. By this time, £43 million had already been expended on the project and out of which £20 million was confirmed wasted. RISP was meant to accomplish integration across the health region. The failure of the project was attributed to the ambiguous definition of the scope which resulted in difficulties in handling and budgeting the expenditure of the project [3].

In this paper, an empirical study and comparison of two models on NASA dataset [4]. K-Nearest Neighbour and Constructive COst MOdel II (COCOMO II) are the methods which are utilised in this work. These methods have seen an explosion of interest over years and hence it is important to analyse the performance of these methods. These methods have been analysed on datasets collected from 90 projects.

The paper is organized as: Section 2 summarizes the related work. Section 3 explains the research background, i.e, describes the dataset used for the estimation of effort and also explains various performance measures. Section 4 presents the research methodology followed in this paper. The results of the models estimated for software effort estimation and the comparative analysis are given in Section 5. The paper is concluded in Section 6. Finally, the future research is discussed in Section 7.

II. RELATED WORK

Software Effort Estimation is one of the most significant fields in software engineering and has repeatedly drawn the attention of researchers and practitioners towards addressing the on-going problem of inaccurate estimates in software development. Software effort estimation requires high accuracy, but it is difficult to achieve accurate estimates. Software effort estimation also plays a key role in determining cost of a software project. Software cost estimation includes the generation of the effort estimates and project duration in order to compute cost required to develop software project. There are various Software Estimation Techniques which fall in the following categories: Expert Judgment, Algorithmic Models, Machine Learning, Empirical techniques, Regression techniques and Theory-based techniques.

Enhancing the accuracy of effort estimation remains an intricate problem because it is difficult to deduce which technique produces more accurate estimation on which dataset [5]. According to software effort estimation technique survey, it is concluded that there is no single technique that can lead

us to unambiguous results [6].

Several models have been proposed for Software effort estimation, e.g., Slim, Cocomo, Estimacs, Function Point Analysis (FPA), Spans, Costar etc. In any case, none of the models proposed have been outstandingly successful in precisely predicting the effort required to develop a particular software product [7].

Many machine learning methods have been the subject of comparison to seeking an accurate estimation model for software development effort [8]. Several studies comparing techniques have been conducted for software effort estimation. From 17 different organisations dataset of 299 software projects were used from which they were divided randomly into 249 training cases and 50 test cases [9]. Desharnais compared using function points Analysis (FPA) with Artificial Neural Networks (ANN), Case-Based Reasoning (CBR) and Regression Models. Desharnais concluded that Artificial Neural Networks model performs more effectively than the other techniques.

Jain and Malhotra (2011) compared Linear regression, Support Vector Machine (SVM), Artificial Neural Network (ANN), Decision tree and Bagging using 499 projects. The result showed that Decision tree is the best among the other techniques at predicting effort required to develop a software.

Over the years, machine learning has been producing good results in numerous fields and like the majority of the effort prediction research, these reports assess the accuracy of effort estimation models. The significant distinction between this report and the current research attempt is that it goes into the comparison of the traditionally utilized algorithmic model of software effort estimation with a popular machine learning technique not currently researched by most.

III. RESEARCH BACKGROUND

A. Feature Sub Selection Method

The data we have used is obtained from Promise data repository. The NASA dataset originally comprises of 93 instances, however, 90 of these instances are chosen after disregarding unusable instances. The disregarded instances are that of the organic software projects, given the number of instances, they would not be sufficient to implement the proposed technique. Each of this projects are described by 15 effort multipliers and are measured on the scale of six categories ranging from very low to extra high. The 90 project chosen consist of 69 semi-detached and 21 embedded software projects.

B. Machine Learning Method

After data refinement, the projects are then divided into training sets and test set on a ratio of 7:3 (Given a dataset of size "B", divided into a training set of size (Y=|Y set| <= B) and the test set of size (X = B – Y=|Y set|). Thus 70% was used for the training set and 20% was used for the test set.

The selected training set of semi-detached software projects are 48 projects and the remaining 21 projects are used as test set. Likewise, the selected training set of embedded software projects are 15 projects and the remaining 6 projects are used as test set. The two techniques (k-NN and COCOMO II) are both implemented in the MATLAB environment [10] and the estimated effort of the test set is generated and compared with the equivalent actual effort in the original dataset to verify the estimation capability of the method. For the k-NN technique, the estimated effort is generated using the Euclidean distance function and then different values of k used to examine which value produces better results while for the COCOMO II technique, effort is estimated using the COCOMO II formula for both the semi-detached and embedded software projects.

C. Performance Measures

The following evaluation criterion has been used to evaluate the estimate capability. Amongst all these stated criterions, the most frequently utilised for performance measure are the PRED (n) and MMRE. Hence, these two measures are used in the comparison of our results with the results of preceding researches.

1) *Mean Magnitude of Relative Error (MMRE)* is a measure of the average error given by an estimation system. It is achieved through the average of the Magnitude of Relative Error (MRE), MRE is calculated as the summation of the absolute difference between the actual effort and the predicted effort divided by the actual effort.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|AE_i - PE_i|}{AE_i}$$

Where, PE_i is the predicted effort achieved for the i^{th} test data:

AE_i is the actual effort collected for the i^{th} test data

n is the total number of projects in the test set.

2) *Root Mean Squared Error (RMSE)* is a regularly utilised measure of differences between estimated value of the model and the actual perceived value. It is obtained through the square root of the Mean Square Error (MSE), MSE is calculated as the squared difference between the actual effort and the predicted effort.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(AE_i - PE_i)^2}$$

Where, PE_i is the predicted effort achieved for the i^{th} test data:

AE_i is the actual effort collected for the i^{th} test data

n is the total number of projects in the test set.

3) *Mean Absolute Error (MAE)* is the measure of how far the predicted values are from actual values. It is calculated as the mean of the absolute errors between predicted and actual effort.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|AE_i - PE_i|$$

Where, PE_i is the predicted effort achieved for the i^{th} test

data:

AE_i is the actual effort collected for the i[th] test data

n is the total number of projects in the test set.

4) *Relative Absolute Error (RAE)* is the total absolute error made relative to what the error would have been if the estimate just had been the average of the actual values. It is obtained through the summation of the absolute difference between actual and predicted effort divided by the summation of the absolute difference between actual and mean of actual.

$$RAE = \frac{\sum_{i=1}^{n} |AE_i - PE_i|}{\sum_{i=1}^{n} |AE_i - AE_m|}$$

Where, PE_i is the predicted effort achieved for the i[th] test data:

AE_i is the actual effort collected for the i[th] test data

AE_m is the average of the actual effort collected for the i[th] test data

n is the total number of projects in the test set.

5) *Root Relative Squared Error (RRSE)* is calculated by dividing the RMSE by summation of the squared difference between actual values and mean of actual values. Therefore, the smaller the values the better.

$$RRSE = \sqrt{\frac{\sum_{i=1}^{n}(AE_i - PE_i)^2}{\sum_{i=1}^{n}(AE_i - AE_m)^2}}$$

Where, PE_i is the predicted effort achieved for the i[th] test data:

AE_i is the actual effort collected for the i[th] test data

AE_m is the average of the actual effort collected for the i[th] test data

n is the total number of projects in the test set.

6) *Correlation Coefficient* is the statistical measures of the strength and direction of a relationship between two variables. It indicates the strength of the relationship. The higher the value of correlation coefficient, the stronger the relationship.

$$Cov(AE, PE) = \frac{1}{n}\sum_{i=1}^{n}(AE_j - \overline{AE})(PE_j - \overline{PE})$$

$$\sigma_a = \sqrt{\frac{\sum_{i=1}^{n}(AE_j - \overline{AE})^2}{n}} \qquad \sigma_p = \sqrt{\frac{\sum_{i=1}^{n}(PE_j - \overline{PE})^2}{n}}$$

$$C_j = \frac{Cov(AE, PE)}{\sigma_a . \sigma_p}$$

Where, PE_i is the predicted effort achieved for the i[th] test data:

AE_i is the actual effort collected for the i[th] test data

$\overline{AE}$ is the mean of the actual effort collected for the i[th] test data

$\overline{PE}$ is the mean of the predicted effort collected for the i[th] test data

n is the total number of projects in the test set.

7) *Prediction Accuracy (PRED (n))* is an indicator of the percentage of estimates that are within n% of the actual values. It is obtained from the relative error which is the absolute difference between the actual and predicted effort. It is expressed as the ratio of the test data with relative error (RE) less than or equal to x percent to the total number of projects in the test set. Hence, the larger the value of PRED (n), the better it is. n should be 25% and a good prediction system ought to attain this accuracy level 75% of the time. [11]

$$PRED(x) = \frac{t}{n}$$

Where, t is the value of relative error (RE) where the test data has:

less than or equal to x.

n is the total number of projects in the test set.

IV. Research Methodology

In this paper, one machine learning technique and one algorithmic model is used in order to predict effort. K-Nearest Neighbour and COCOMO II have seen an explosion of interest over the years, and have successfully been applied in various areas.

A. K-Nearest Neighbour

K-Nearest Neighbour (k-NN) algorithm is a non-parametric machine learning method for classification that predicts objects class by classifying objects centred on the k nearest training examples in the feature space [12]. K-Nearest Neighbour is recognised for its ability to produce good results in clinical outcome prediction such as Cancer prediction.

The k-nearest neighbour (k-NN) classifies new cases with previously stored available cases on the basis of similarity measure (e.g., Distance functions). It is a type of instance-based learning where the function is only approximated locally and all calculation is postponed until classification. The k-Nearest Neighbour (k-NN) gathers historical data known as the training data set, and utilises this data collected to make estimates for new test data, and then, the k-nearest data of the training dataset are achieved. Based on the data attribute of the nearest records, an estimate is made for the new data.

The k-Nearest Neighbour (k-NN) classification algorithm expands this procedure by utilising a predefined number ($k \geq 1$) of the nearest training instances as opposed to utilising only a single instance. *k* is a user-defined constant of positive integers, usually small. For instance, in a self-organizing map (SOM), all nodes are representatives of a cluster of similar points, irrespective of their density in the original training data. K-Nearest Neighbour(k-NN) can then be applied.

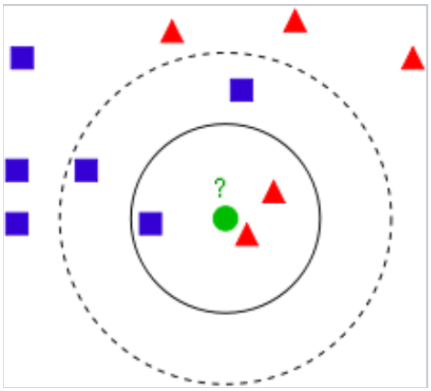

Fig. 1. Example of k-NN classification [13].

The test sample (green circle) ought to be classified either to the top class of blue squares or to the lower class of red triangles. When k = 3 (solid line circle) it is allotted to the red triangles because there are 2 triangles and just 1 square inside the inner circle. In the event that k = 5 (dashed line circle) it is allotted to the blue squares (3 squares versus 2 triangles inside the external circle) [13].

In k-Nearest Neighbour (k-NN) classification (Fig. 1), the estimated label is dictated by the voting for the nearest neighbours to the test label, that is, the most common label in the set of the chosen k instances is returned. The quality of the estimation relies upon the distance measure.

The generally used distance function is the Euclidean Distance.

$$Euclidean = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Where, k is the user-defined constant

i is the number of instances

x and y are the vectors of each instance.

B. COCOMO II Model

COnstructive COst MOdel II (COCOMO II) is an estimation method that enables one to predict cost, schedule and effort when planning the development of a new project. The COCOMO techniques represents a model-based, data driven, parametric method that executes a fixed model method [14]. Thus, COCOMO produces a fixed estimation model which has been formed on organisational project data by utilising statistical regression, which signifies parametric and data driven techniques. COCOMO II is the latest version of the original COCOMO also known as COCOMO 81.

In 1981, the original COCOMO model was created by Dr. Barry Boehm using a multiple regression analysis. This was derived from the evaluation and scrutiny of 63 software development projects [15]. The use of the effort estimation equation to predict the number of person-months or person-hours needed to develop a project is the most essential calculation in the COCOMO model [15]. The model applies to three types of software projects (Table 1) [16]:

Organic projects – These are projects that consist of relatively small teams with lots of experience with less stringent requirements.

Semi-detached projects – These are projects that consist of medium teams with diversified experience working with a combination of stringent and less stringent requirements.

Embedded projects – These are projects developed with stringent requirements and team, that have little experience in the project area. It is also a mixture of organic and semi-detached projects.

TABLE I. TYPES OF SOFTWARE PROJECT [16]

Software project	**a**	**b**
Organic	2.4	1.05
Semi-Detached	3.0	1.12
Embedded	3.6	1.21

The original COCOMO model equation for computing effort is as follows:

$$Effort(E) = a \times (KLOC)^{b} \times EAF$$

Where, KLOC is the estimated size (number of lines of code) of the software project (expressed in thousands).

The coefficients a, b are dependent on which type of software project is being developed.

EAF (Effort Adjustment Factor) is the product of all the effort drivers. Effort is calculated in person months and it is a function of development criterion productivity, some effort drivers, and software size.

V. ANALYSIS RESULT

A. Model Prediction Result

The k-Nearest Neighbour (k-NN) and COCOMO II techniques have been used for estimating the efforts required to develop a software project using NASA dataset. Effort was estimated for both semi-detached and embedded software project. Seven performance measures have been utilised to compare the results gotten from these models. These measures are Mean Magnitude-Relative-Error (MMRE), Root-Mean Square-Error (RMSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), Relative Root Square Error (RRSE), Correlation Coefficient and PRED (25). The technique holding low values of MMRE, RMSE, MAE, RAE, and RRSE and high values of PRED (25) and correlation coefficient is considered to be the best among others.

TABLE II. RESULT

Performance Measures	Semi-Detached Software Project		Embedded Software Project	
	COCOMO II	k- Nearest Neighbour (k-NN)	COCOMO II	k- Nearest Neighbour (k-NN)
Mean Magnitude Relative Error (MMRE)	1.07	**0.54**	**1.70**	7.84
Root mean squared error (RMSE)	20266.30	**3189.30**	**1501.24**	5961.46
Mean absolute error (MAE)	1198.20	**393.70**	**429.36**	1378.92
Relative absolute error (RAE)	1.98	**0.65**	**0.20**	0.65
Root relative squared error (RRSE)	4.47	**0.70**	**0.21**	0.84
Correlation coefficient	**0.91**	0.76	**0.998**	0.98
PRED (25) %	23.81	**28.57**	**33.33**	16.67

The plots in Fig. 2 to 5 display the results of k-Nearest Neighbour (k-NN) and COCOMO II on semi-detached and embedded software project. In these plots, the purple curve represents the estimated values and the orange curve represents the actual values. The closer, the actual and estimated value curves are the lower, the error and the better the technique.

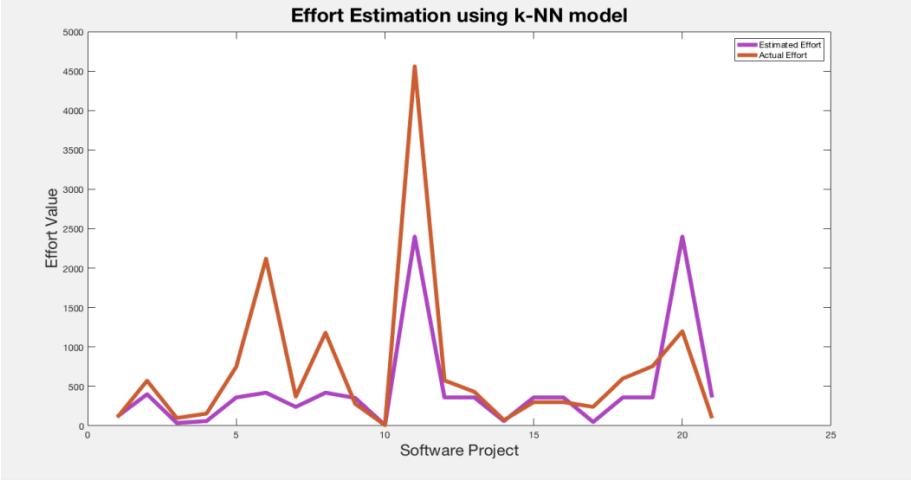

Fig. 2. Result using the k-Nearest Neighbour (k-NN) for Semi-Detached software project.

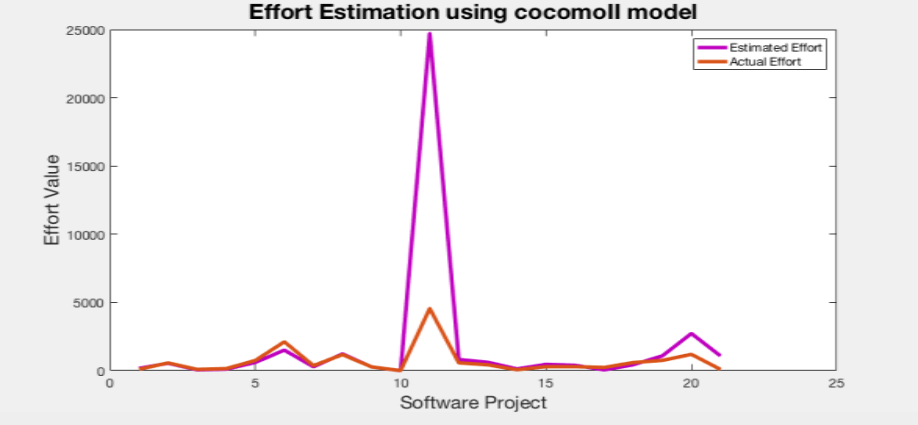

Fig. 3. Result using the COCOMO II for Semi-Detached software project.

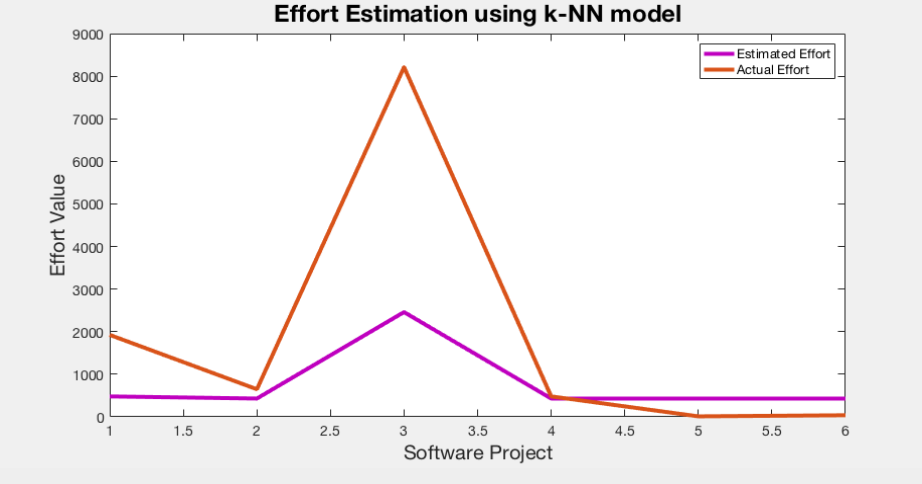

Fig. 4. Result using the k-Nearest Neighbour (k-NN) for Embedded software project.

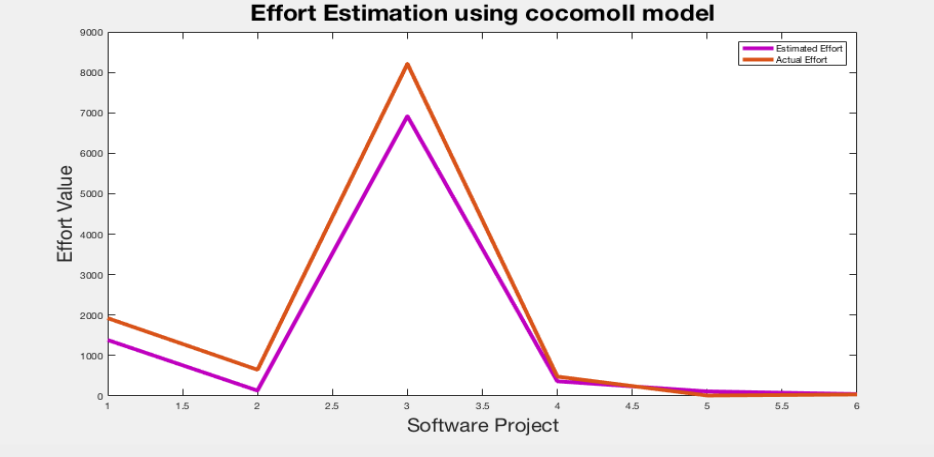

Fig. 5. Result using the COCOMO II for Embedded software project.

VI. CONCLUSION

In this research, the k-Nearest Neighbour (k-NN) and COCOMO II techniques have been used in predicting software development effort. Results were obtained using the NASA dataset acquired from Promise software engineering repository and are displayed in Table 2 above. The two techniques (k- NN and COCOMO II) were applied to the semi-detached and embedded software project and the method with lower error and higher accuracy is considered as better suited for estimating effort. There are various estimation techniques, although, no one method is necessarily better or worse than the other. It is difficult to decide which technique is right for which dataset [17].

The embedded software project dataset consists of 15 historical training data and 6 test set while semi-detached software project dataset consists of 48 historical training data and 21 test set. The results show that the k-Nearest Neighbour (k-NN) was the best technique for estimating the effort with MMRE value 0.54 and PRED (25) value 28.57% for the semi-detached software project. Hence, the NULL hypothesis H_0 should be accepted. Although, for the embedded software project the results show COCOMO II to be the best method with MMRE value 1.70 and PRED (25) value 33.33%. Thus, the Alternate hypothesis H_A should be accepted.

This result shows that k-NN is better when applied to the semi-detached software project and worse for embedded software project. This, however, does not signify that the type of software project has any effect on the method utilised. Thus indicating that the nature of data is important and this is true for most machine learning techniques. Small data makes over-fitting harder to avoid which is by far the most common problem in applied machine learning [18], and the outliers could mislead the training process thereby affecting the results too. In comparison to the previous study by Jain and Malhotra (2011), several machine learning methods such as decision tree, linear regression, bagging etc. were analysed on a larger dataset and produced good results.

The plot in Fig. 3 above shows an extreme overestimation of the semi-detached project which is with regards to the high rating scale of the effort driver in the dataset.

The first research question was “Which of the two techniques (k-NN and COCOMO II) produces better estimates?” Considering this question through the result obtained from both software projects, this question can, in

fact, have multiple answers: the k-NN technique performs better when there is a substantial amount of data. The lack of data makes k-NN unsuitable for use and as shown in Fig. 5 above, the COCOMO II model produces better estimates when analysed on the embedded project dataset because the nature of data has no effect on the algorithmic model.

The second research question was "Which of the two techniques (k-NN and COCOMO II) has stronger relationship between its estimated and actual value?" The result in Table 2 above shows that COCOMO II model has a stronger relationship with a correlation coefficient of 0.91 for semi-detached project and 0.998 for the embedded project.

Software practitioners and researchers may apply the k-Nearest Neighbour (k-NN) method for effort estimation where there is adequate data.

VII. FUTURE RESEARCH

Researchers have examined various machine learning techniques which are producing good results in different domains. However, the proposed technique has not received adequate attention in the field of software effort estimation. It would be useful to compare this k-Nearest Neighbour (k-NN) technique with other techniques such as Artificial Neural Network (ANN), Decision Tree, Neuro- Fuzzy (NF) etc. being examined by numerous researchers.

For future research, this methodology can further be explored on some large datasets to improve the validity of the produced results. Also, the performance of the proposed estimation method could be enhanced by making further modifications regarding small data by utilising the "leave one out" cross validation process, boot strapping and many more resampling techniques.

REFERENCES

[1] P. Naur, and B. Randell, "Software Engineering." 1969.

[2] I. Sommerville, "Software engineering." 9th ed. Addison-Wesley. 2011.

[3] M. Jeffcott, and C. Johnson, "The Use of a Formalised Risk Model in NHS Information System Development." 2002.

[4] Promise. Available: http://promisedata.org/repository/.

[5] A. Jain, and R. Malhotra, "Software Effort Prediction using Statistical and Machine Learning Methods." IJACSA, Vol.2, pp. 145-152, 2011.

[6] H. Rastogi, and S. Dhankhar "A survey on Software Effort Estimation Techniques" IEEE Xplore Document. *5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, pp.826-830, 2014.

[7] C. Kemerer, "An empirical validation of software cost estimation models." Communications of the ACM, Vol.30, pp.416-429, 1987.

[8] Y. Kim, and K. Lee, "A Comparison of Techniques for Software Development Effort Estimating." 2017.

[9] J. Desharnais, G. Wittig and G. Finnie, "A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models." *Journal of Systems and Software*, vol.39, pp.281-289, 1997.

[10] Matlab.Available: https://uk.mathworks.com/products/matlab-online.html

[11] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Using Evolutionary Based Approaches to Estimate Software Development Effort." 2010.

[12] K. Conley, and D. Perry, "A Recommendation Engine for Picking Heroes in Dota 2."

[13] S. Ananthi, and D. Sathyabama, "Spam Filtering Using K-NN." Journal of Computer Applications, Vol.2, pp.20-23, 2009.

[14] A. Trendowicz, and R. Jeffery, "Software Project Effort Estimation." 1st ed. 2014.

[15] B. Boehm, "Software Engineering Economics." 1st ed. New York: Prentice-Hall Inc. pp.200- 217, 1981.

[16] S. Sehra, J. Kaur, and S. Sehra, "Effect of Data Preprocessing on Software Effort Estimation." International Journal of Computer Applications, Vol.69, pp.33-36, 2013.

[17] P. Rijwani, D. Santani, and S. Jain. "Software Effort Estimation: A Comparison Based Perspective." IJAIEM Vol.3, pp.18-29, 2014.

[18] J. Brownlee, "How to Identify Outliers in your Data -Machine Learning Mastery." 2013.

AUTHORS PROFILE

Razak Olu-Ajayi: He is a student at the Department of Computer Science, University of Hertfordshire. He received his bachelor's degree in Computer Science from Babcock University, Ogun, Nigeria. He is currently studying his masters in software engineering at the University of Hertfordshire, Hatfield, United Kingdom. His research interests are in software Estimation, improving software quality, statistical and adaptive prediction models.