Effort Estimation in Agile Software Development Using Autoencoders

Eduardo Rodríguez Sánchez, Eduardo Vázquez Santacruz, and Humberto Cervantes Maceda
PCyTI, Departamento de Ing. Eléctrica
Universidad Autónoma Metropolitana, Ciudad de México.
E-mail: erodsmx@gmail.com, evazquez.santacruz@izt.uam.mx, hcm@xanum.uam.mx

2023 12th International Conference On Software Process Improvement (CIMPS) | 979-8-3503-5856-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/CIMPS61323.2023.10528839

***Abstract*—Effort estimation is important to correctly plan the use of resources in a software project. When adopting Agile in IT, business value is raised in both performance and quality. A gap in agile effort estimation is the lack of research combining software engineering models and deep learning techniques. During the planning phase the team involved makes an approximate estimation of time and cost based on artifacts and requirements obtained from initial interviews with clients and stakeholders.**

This paper aims to contribute with a hybrid effort estimation model that uses story points which measure the amount of effort needed to accomplish the project, team velocity which measures how many units of effort the team completes in a typical Sprint, and category size labels of effort, time and cost in order to estimate completion time and total cost of a project developed with agile methods like Scrum. The machine learning techniques used to implement the project are neural networks such as autoencoders and different variations of it. The learning capabilities are assessed through 10-Fold cross validation and the estimates are compared with the original dataset and the results obtained from literature. This research uses 21 projects developed by six software houses, a set of 42 noisy data is used for training created using data augmentation technique. Each project has two dependent variables that the algorithm tries to estimate and they are completion time measured in days and total cost valued in Pakistan rupees.

The proposed approach compares the use of the original data as input versus the original data with the addition of category size labels. The main idea is that every project has three main features that are scope, time and cost. Since the current work is based on historical data the scope is always fixed and a single project can be estimated according to a hypothetical time or cost which can be small, medium or large.

***Keywords*-Agile, Effort, Estimation, Time, Cost, Neural Network, Autoencoders.**

I. Introduction

Enterprises and organizations need to respond quickly to customers and stakeholders' needs, adapt to changes and keep it over static documents. Scrum is an agile method created by Jeff Sutherland and Ken Schwaber as a more reliable, effective and faster way to create software. It came in a time where software developments were still done with the waterfall method. During the last fifteen years, Agile methods have been spreading across organizations according to the States of Agile Report. This is leading the way through the pandemic, distributed work and digital transformation. Organizations are assessing customer satisfaction and business value so it's very important to get builds on time and stay on budget ensuring the scope, time and cost of a project [10], [13], [16].

To ensure delivery times and avoid product cost overruns, it is essential to have a model that allows estimating the effort required to complete the development. So this research project presents an effort estimation model based on Zia et al. work [8] and a deep learning architecture in order to estimate completion time and total cost of a project that uses Agile methods like Scrum. In section II a synthesis of the works related to the effort in software development is presented. Then in section III the proposed approach is broken down and the main strategy is explained showing the different configurations of the deep learning models. Results are grouped in section IV showing the differences between configurations, the use of labeling and the regression lines of the estimates, also a comparison with the results from literature is shown. Finally in section V conclusion and future work is presented highlighting the usefulness of the strategy used.

II. Related Work

Effort estimation has been around since 1978 with one of the first models created by Putnam called SLIM and followed by a model based on function points proposed by Albrecht. Another popular model came later in 1981 presented by Boehm and called the Constructive Cost Model which uses data from historical projects and fits a regression formula. As years went by, this model provided researchers a valuable dataset for building different approaches based on machine learning techniques to provide better estimations [8], [15], [25], [26].

Several works have used public dataset combined with machine learning techniques in order to do effort estimation. Some of the techniques include support vector machines, KNN, decision trees, neural networks, expert systems, among others [20], [21], [22], [24], [23], [25], [27].

In 2019 Huynh Thai Hoc et al. [26] reviewed existing Software Project Effort Estimation (SPEE) exhaustively by exploring Regression Models for modern SPEEs. The work groups several techniques like expert judgment, analogy-based, Price-to-win, Bottom- up and Top-up, Wideband Delphi, Planning Poker and algorithmic ones like FPA, object point, COCOMO and use case point [26].

Manju Vyas et al. [10] captures the main contributions to effort estimation, The study makes a synthesis of the main effort estimation proposals classifying models in four categories: Expert judgment, analogy methods, algorithmic models, and machine learning models [10].

Agile software development came in the early 21st century and in 2002 Grenning proposed the use of Planning Poker as an estimation technique, later in 2005 M. Cohn suggested that this method was suitable for agile projects. During the last ten years there have been other proposals that set agile development as the main subject. Marta Fernandez et al. [15] did a systematic literature review on effort estimation in agile software development showing that the most used agile methods are Scrum XP, TDD, Agile Unified Process, Kanban and Distributed Agile Software Development. From data-based estimation methods, some of the most frequently used are Random Forest, Decision Tree, SVM, KNN, Stochastic Gradient Boosting, Naive Bayes and Neural Networks [1], [3]–[5], [15].

E. Coelho and A. Basu [9] highlights the fact that in agile development story points are estimates of the work to do and velocity is a measurement of the progress per sprint, both important features of the model proposed in this work. In 2014 Rashmi Popli and Naresh Chauhan [13] calculated cost, effort and time needed for a small and medium size project using a story point approach.

S. Garg and D. Gupta [16] suggest in 2015 a model based on Principal Component Analysis that identifies the factors affecting development costs and then uses a constraint solving approach to satisfy the criteria imposed by agile manifesto. In the same year Atef Tayh Raslan et al. [4] aims to improve the effort estimation accuracy in software project development by applying the fuzzy logic and Story Points Estimation Model (SPEM) on agile software development.

E. Scott, & D. Pfahl [5] explore a predictive model that uses developers' features to assign story points to issue reports so that it can be used for novice developers during a planning poker session. Authors use a supervised approach that includes Support Vector Machines (SVM) [5].

The effort in the maintenance phase is studied by Jitender Choudhari and Ugrasen Suman [11]. The model used in this study is applicable only for agile and extreme programming based maintenance environment. The technique uses story points to calculate the volume of maintenance and value adjustment factors that are affecting story points for effort estimation [11].

O. Malgonde & K. Chari [28] developed a model for story-effort prediction using variables that are readily available when a story is created. This approach has three main components: first a predictive model to predict story effort, second an ensemble-based approach to aggregate predictions, and third an optimization model to optimize available effort [28].

Also in 2019 Morakot Choetkiertikul et al. [7] proposed a prediction model for estimating story points based on two powerful deep learning architectures: long short-term memory and recurrent highway network (RHN). During experiments authors compared RHN against Random Forest, SVM, Automatically Transformed Linear Model and Linear Regression. The model is a fully end-to-end prediction system for estimating story points, removing the users from manually designing features from the textual description of issues [7].

Another example of estimating at user story level is the research made by M. Durán et al. [18]. The purpose of this research is to provide development teams without experience or without historical data, a method to estimate the complexity of user stories through a model focused on the human aspects of developers [18].

M. Gultekin & O. Kalipsiz [19] state that the main objective of the study is to objectively and accurately estimate the effort when using the Scrum methodology. A dynamic effort estimation model is developed by using regression-based machine learning algorithms such as Support Vector Regression, Random Forest Regression, Multilayer Perceptron and Gradient Boosting algorithm which outperforms in accuracy. The study uses both arithmetic and fibonacci series as a scoring system showing that fibonacci got better accuracy on estimation [19].

Within the agile effort estimation models we can find the work proposed by Zia et al. [8] which is taken for the current research. This work is intended to calculate completion time and total cost for the agile software project. It uses very basic concepts and equations like story size, complexity of the requirements of the story and or its technical complexity, the velocity team, friction factors and dynamic forces that might affect the project. Three research projects take Zia's work as a base for applying more machine learning techniques [8].

Aditi Panda et al. [1] uses four types of neural networks from which Cascade-Correlation Neural Network achieves the best accuracy and lowest error of them all. Ch. Prasada Rao et al [3] uses three types of neural networks and the one with the best results is a Generalized Regression Neural Network. S. M. Satapathy, S. K. Rath [6] works with decision trees, random forest and Stochastic Gradient Boosting, getting the best results from SGB technique. So our research has to deal with three techniques, Cascade-Correlation Neural Network, Generalized Regression Neural Network and Stochastic Gradient Boosting. The proposed approach tries to match the results from literature proving that neural networks are reliable techniques to improve accuracy of estimates.

III. Proposed approach

Our work uses the effort estimation model proposed in [8]. The model is based on a story points approach having several features to estimate the time and cost of a project such as the total effort to complete the project, the team

velocity, deceleration due to friction factors and dynamic forces, the sprint size, the working days in a month and the team salary.

Our research project proposes a hybrid model which combines an algorithmic model and a machine learning model based on neural networks. The difference between the works presented in the literature [8] and our project is the incorporation of deep learning neural networks that have the ability to compress information and make estimates in the same model. So there is no need to use different techniques to do Principal Component Analysis and pick the most relevant features and then make predictions based on these features.

The machine learning algorithm takes as input the whole dataset and adds category size labels in order to group projects by size of effort, time and cost. Information is compressed by an autoencoder and then estimates of time and cost are done. There are three architectures tested in the experiments. First a traditional autoencoder which has two blocks, an encoder and a decoder, the second architecture only tests the encoder block and a third approach uses the encoder block and adds additional feed-forward layers to test if the deeper network improves the accuracy. Finally, the AdaBoost ensemble method is used with the encoder architecture to test whether or not the accuracy improves. The final results are averaged in a bagging method so the final ensemble is made of four independent techniques.

The models are trained using 10-Fold cross-validation in order to assess the learning capabilities of the algorithms. Because of the dataset's small size, a technique called data augmentation [29] is used to duplicate samples by adding small amounts of noise to effort and velocity resulting in 42 noisy values for training and 21 real values for testing. This helps to address overfitting when training the model and with cross validation we end up with 20 trained models since a repeated k-Fold is executed twice resulting in 20 partitions. In k-Fold data is divided into k subsets of approximately equal size and each partition is used to train the model. With validation the model is checked after each training step in order to help determine if overfitting is happening.

Once models are trained, predictions of the 21 real projects are done using the real values of the dataset. All predictions are collected and averaged to get a final estimate and compute accuracy, coefficient of determination, mean squared error, root of mean squared error, mean relative error and explained variance. Results of the deep neural nets are compared to those in literature via accuracy and relative error. The general strategy applied is shown in the flowchart of Fig. 1. The first column on the left of the flowchart is part of data preprocessing, the second column on the right belongs to training and model evaluation.

Regarding the deep learning algorithms, three architectures are used to perform the estimates. The first one is an autoencoder whose input layer is equal to the nine features, each layer is reduced in three neurons so the encoder has nine, six and three neurons. The decoder increases by 3 neurons having two layers of six and nine neurons as shown in Fig. 2a. The second architecture uses only the first block, an encoder with three layers going from nine to six and three neurons as shown in Fig. 2b. A third architecture adds two extra layers of two neurons to the encoder to get a deeper network as shown in Fig. 2c.

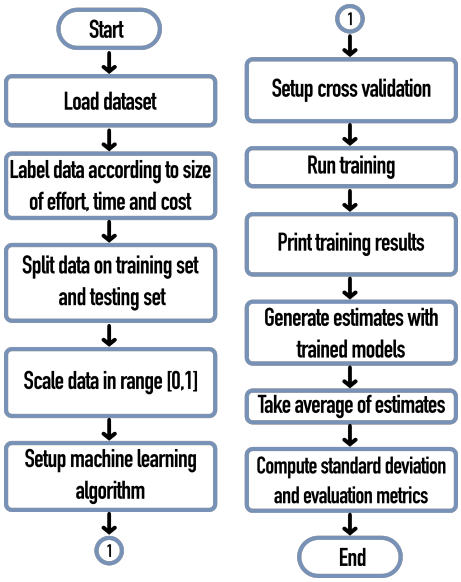
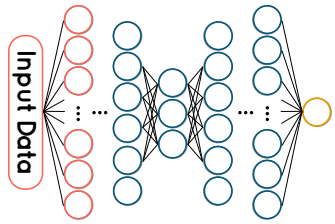


Figure 1: Project strategy to generate time and cost estimates.

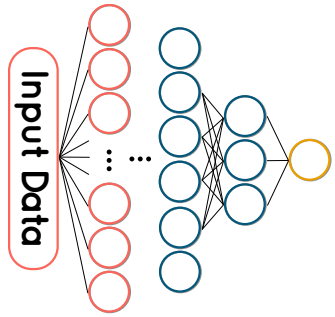
IV. Results

All deep learning models share a couple of features which are the basic setup. The optimization algorithm for training is Adam, optimizers are methods used to change the weights and learning rate in order to reduce the function loss. Stochastic Gradient Descent is the most used optimization algorithm and Adam is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. This optimizer is good for problems that are large in terms of parameters or data. The activation function for all neurons is the Tanh function, the learning rate is set to the default value of 0.001. The loss function used to evaluate each epoch is the Mean Squared Error. Training for all models is set to a maximum of 3000 epochs and every model executes two cases changing the input data in order to show that adding labels improves the accuracy of estimates.

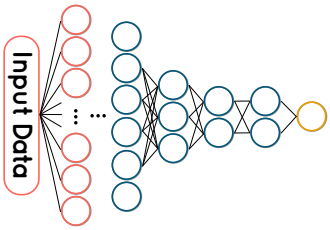
In Table I the results of Autoencoder are grouped. The gap between the values for training and validation are not big so it ensures there is neither underfitting nor overfitting. The autoencoder gets better results when using labels for both time and cost. The coefficient of determination improved with an increase of 0.0165 in time estimates and 0.0427



(a) Autoencoder



(b) Encoder



(c) EncoderMLP

Figure 2: All network architectures used to make predictions are feed-forward type.

increase in cost estimates when testing with real data values. In both cases a decrease for the RMSE shows that labels improve the accuracy of predictions.

In Table II results for the encoder architecture are shown. In this case only time estimates improved when using labels and the accuracy and R^2 decreased when estimating total cost. Compared to the Autoencoder, the second architecture got similar results and the model is half the size of the autoencoder which is good for architecture simplification.

Finally the last architecture which consisted of an encoder with two extra layers had good results for completion time but it got worse results for total cost. These values are shown in Table III and the change in architecture did not improve the estimates at all, so it means that a deeper network does not mean better results.

Once the three models are tested, only the one with the simplest architecture is trained using the AdaBoost ensemble method. The ensemble improved R^2 , going from 0.9915 to 0.9928 for completion time and 0.9143 to 0.9856 for total cost. The error for the single encoder and the ensemble stayed at two days but the total cost error decreased from

	Parameter	Model Autoencoder			
Setup	Optimizer	Adam			
	Activation	Tanh			
	Hidden Layers	4			
	Input Dimension	9			
	Neurons per Layer	Reduction and increase in 3 neurons			
	Learning rate	0.001			
	Loss Function	Mean Squared Error (MSE)			
	k-Fold Cross Validation	RepeatedKFold with 10-Fold and n_repeats=2			
	Data Normalizing	MinMaxScaler with values between (0,1)			
	Input	Dataset	Dataset + Labels	Dataset	Dataset + Labels
	Output	Time		Cost	
Training	MSE	0.0119	0.0018	0.0139	0.0055
	RMSE	0.0652	0.0349	0.0900	0.0629
	Coefficient of determination (R^2)	0.8362	0.9753	0.7656	0.9089
	Explained Variance	0.8571	0.9755	0.7929	0.9094
Validation	MSE	0.0230	0.0050	0.0176	0.0066
	RMSE	0.0869	0.0592	0.0911	0.0639
	R^2	0.6997	0.8875	0.4999	0.8095
	Explained Variance	0.8288	0.9238	0.7409	0.8409
Test	Accuracy (%)	**93.86**	**95.32**	**91.89**	**91.77**
	R^2	**0.9723**	**0.9888**	**0.9059**	**0.9487**
	MRE	0.0614	0.0468	0.0811	0.0823
	MSE	18.00	7.29	44,565,746,289	24,329,148,553
	RMSE	4.24	2.70	211,106	155,978
	Explained Variance	0.9759	0.9890	0.9108	0.9490
Regularization	Dropout	No			
	Gaussian Noise	stddev=0.00001			
	Num of noise layers	2			
Maximum Training Epochs		3000			
Average Training Epochs		1204	1823	519	904

Table I: Training and testing results of Autoencoder.

	Parameter	Model Encoder			
Setup	Optimizer	Adam			
	Activation	Tanh			
	Hidden Layers	2			
	Input Dimension	9			
	Neurons per Layer	Reduction in 3 neurons			
	Learning rate	0.001			
	Loss Function	Mean Squared Error (MSE)			
	k-Fold Cross Validation	RepeatedKFold with 10-Fold and n_repeats=2			
	Data Normalizing	MinMaxScaler with values between (0,1)			
	Input	Dataset	Dataset + Labels	Dataset	Dataset + Labels
	Output	Time		Cost	
Training	MSE	0.0013	0.0010	0.0051	0.0110
	RMSE	0.0330	0.0293	0.0654	0.0865
	Coefficient of determination (R^2)	0.9828	0.9861	0.9127	0.8159
	Explained Variance	0.9830	0.9863	0.9130	0.8231
Validation	MSE	0.0030	0.0038	0.0083	0.0105
	RMSE	0.0459	0.0496	0.0712	0.0816
	R^2	0.9413	0.9109	0.7889	0.5684
	Explained Variance	0.9612	0.9208	0.8821	0.6718
Test	Accuracy (%)	**95.77**	**96.04**	**91.93**	**89.03**
	R^2	**0.9901**	**0.9915**	**0.9367**	**0.9143**
	MRE	0.0423	0.0396	0.0807	0.1097
	MSE	6.43	5.52	29,982,176,142	40,618,680,183
	RMSE	2.54	2.35	173,154	201,541
	Explained Variance	0.9904	0.9917	0.9367	0.9164
Regularization	Dropout	No			
	Gaussian Noise	stddev=0.00001			
	Num of noise layers	2			
Maximum Training Epochs		3000			
Average Training Epochs		2014	2551	743	963

Table II: Training and testing results of Encoder.

201,541 rupees to 82,475 rupees which is a significant improvement. The summary of evaluation metrics is shown in Table IV and it is clear that the highest coefficient of determination and the lowest RMSE.

Parameter		Model Encoder + MLP			
Setup	Optimizer	Adam			
	Activation	Tanh			
	Hidden Layers	4			
	Input Dimension	9			
	Neurons per Layer	Reduction in 3 neurons			
	Learning rate	0.001			
	Loss Function	Mean Squared Error (MSE)			
	k-Fold Cross Validation	RepeatedKFold with 10-Fold and n_repeats=2			
	Data Normalizing	MinMaxScaler with values between (0,1)			
	Input	Dataset	Dataset + Labels	Dataset	Dataset + Labels
	Output	Time		Cost	
Training	MSE	0.0049	0.0013	0.0094	0.0100
	RMSE	0.0414	0.0320	0.0879	0.0870
	Coefficient of determination (R^2)	0.9364	0.9828	0.8308	0.8332
	Explained Variance	0.9394	0.9829	0.8366	0.8358
Validation	MSE	0.0056	0.0038	0.0171	0.0118
	RMSE	0.0542	0.0539	0.0871	0.0830
	R^2	0.9011	0.9038	0.7344	0.7424
	Explained Variance	0.9117	0.9237	0.8065	0.8038
Test	Accuracy (%)	**95.34**	**95.96**	**91.21**	**89.67**
	R^2	**0.9873**	**0.9901**	**0.8900**	**0.8973**
	MRE	0.0466	0.0404	0.0879	0.1033
	MSE	8.24	6.43	52,142,108,589	48,671,189,953
	RMSE	2.87	2.54	228,346	220,615
	Explained Variance	0.9886	0.9904	0.8920	0.8992
Regularization	Dropout	No			
	Gaussian Noise	stddev=0.00001			
	Num of noise layers	2			
Maximum Training Epochs		3000			
Average Training Epochs		1694	2377	364	773

Table III: Training and testing results of Encoder MLP.

	Algorithm	AE	Encoder	Encoder + MLP	AdaBoost Encoder	Ensemble	AE	Encoder	Encoder + MLP	AdaBoost Encoder	Ensemble
Data	Input	Dataset + Labels									
	Output	Time					Cost				
Results	Accuracy (%)	95.32	96.04	95.96	96.01	**96.10**	91.77	89.03	89.67	93.65	**91.43**
	R^2	0.9888	0.9915	0.9901	0.9928	**0.9920**	0.9487	0.9143	0.8973	0.9856	**0.9460**
	MRE	0.0468	0.0396	0.0404	0.0399	**0.0390**	0.0823	0.1097	0.1033	0.0635	**0.0857**
	RMSE	2.70	2.35	2.54	2.16	**2.28**	155,978	201,541	220,615	82,475	**159,947**
	Explained Variance	0.9890	0.9917	0.9904	0.9937	**0.9922**	0.9490	0.9164	0.8992	0.9859	**0.9470**

Table IV: Summary results for all techniques and the final ensemble.

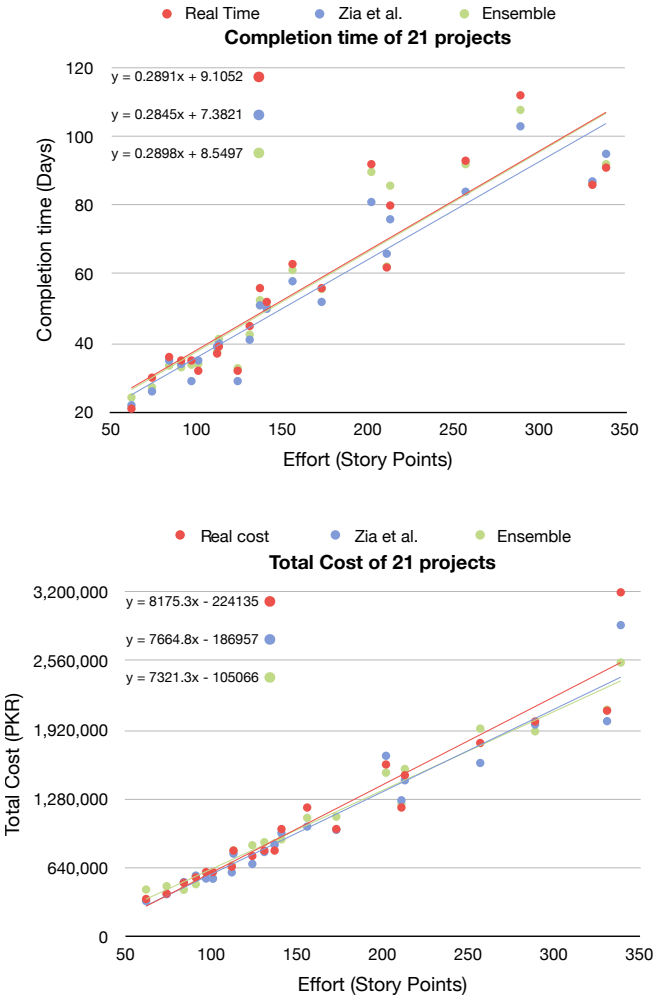


Figure 3: Regression lines for time and cost predictions.

There are a total of four techniques and all of them form a bagging ensemble method that averages the estimates done by the trained models, once the final predictions are obtained evaluation metrics are computed. Even the AdaBoost encoder got the highest results; these are not taken for the final estimates since the objective is to build a machine learning model made of several techniques and not just a single one.

In Fig. 3 comparison between real values, predictions made by the original model [8] and the predictions of the modified model that adds category size labels and machine learning techniques. Red and green lines for completion time are very close which means predictions are closer to real values, however for total cost the predictions look closer to the original model than the real values being the green and blue lines very close.

Once final estimates are computed with their standard deviation as shown in Table V and their evaluation metric are computed, results are compared to those techniques used in literature. It is clear that the biggest difference in completion time does not go beyond six days or one working week with one day. Also for total cost only three projects go over 100,000 rupees and one is over half million rupees.

From all the techniques proposed AdaBoost Encoder outstands the rest of techniques for completion time with an improvement in 16 out of 21 projects and 12 out of 21 projects for total cost. AdaBoost Encoder reduced the difference between real values and predictions. So the best values for completion time were achieved by AdaBoost and the final ensemble got pretty close results too. When comparing cost predictions Adaboost also got the best results and the ensemble was not that close since R^2 values are lower by 0.0396 and RMSE is almost double compared to AdaBoost RMSE.

The Table VI compares the results from literature, including the original model and the proposals using different machine learning techniques. The neural network proposed in this work outstands the Cascade-Correlation Neural Net-

work [1], the Generalized Regression Neural Network [3]. and Decision Trees [6], giving reliable results for time and acceptable results for cost.

No.	ActualTime	Zia et al. (days)	ΔT Zia (days)	Ensemble	ΔT (days)	ActualCost	Zia et al. (PKR)	ΔC Zia (PKR)	Ensemble	ΔC (PKR)	Improved Time?	Improved Cost?
1	63	58	5	61 ± 0	1.8	1,200,000	1,023,207	176,793	1,103,920 ± 29,993	96,081	Yes	Yes
2	92	81	11	90 ± 0	2.3	1,600,000	1,680,664	80,664	1,523,704 ± 81,746	76,297	Yes	Yes
3	56	52	4	56 ± 0	0.3	1,000,000	992,270	7,730	1,115,221 ± 29,662	115,221	Yes	No
4	86	87	1	87 ± 1	0.5	2,100,000	2,002,767	97,233	2,108,781 ± 55,531	8,781	Yes	Yes
5	32	29	3	33 ± 1	0.8	750,000	676,081	73,919	849,893 ± 52,766	99,893	Yes	No
6	91	95	4	92 ± 0	1.0	3,200,000	2,895,133	304,867	2,547,454 ± 260,576	652,546	Yes	No
7	35	29	6	34 ± 0	1.3	600,000	540,114	59,886	596,742 ± 8,698	3,259	Yes	Yes
8	93	84	9	92 ± 0	1.0	1,800,000	1,614,079	185,921	1,933,507 ± 22,829	133,507	Yes	Yes
9	36	35	1	34 ± 1	2.5	500,000	507,265	7,265	435,457 ± 23,051	64,544	No	No
10	62	66	4	62 ± 1	0.3	1,200,000	1,267,180	67,180	1,235,058 ± 11,606	35,058	Yes	Yes
11	45	41	4	43 ± 1	2.5	800,000	786,732	13,268	877,975 ± 32,971	77,975	Yes	No
12	37	39	2	38 ± 1	0.5	650,000	597,143	52,857	659,967 ± 13,172	9,967	Yes	Yes
13	32	35	3	34 ± 0	2.0	600,000	538,495	61,505	542,435 ± 7,119	57,565	Yes	Yes
14	30	26	4	27 ± 0	2.8	400,000	394,546	5,454	469,312 ± 13,028	69,312	Yes	No
15	21	22	1	24 ± 0	3.3	350,000	330,561	19,439	438,016 ± 19,217	88,016	No	No
16	112	103	9	108 ± 1	4.3	2,000,000	1,971,485	28,515	1,907,983 ± 38,732	92,017	Yes	No
17	39	40	1	41 ± 1	2.3	800,000	770,857	29,143	801,499 ± 38,443	1,499	No	Yes
18	52	50	2	51 ± 0	1.3	1,000,000	961,866	38,134	904,354 ± 24,678	95,646	Yes	No
19	80	76	4	86 ± 2	5.8	1,500,000	1,453,032	46,968	1,558,521 ± 53,105	58,521	No	No
20	56	51	5	53 ± 1	3.5	800,000	854,348	54,348	866,998 ± 55,995	66,998	Yes	No
21	35	34	1	33 ± 0	2.0	550,000	567,484	17,484	487,550 ± 9,021	62,450	No	No

Table V: Predictions with their standard deviation.

Author	Estimate	ML Technique	Accuracy (%)	R^2	MMRE
Zia et al.	Time	None	57.14	Not included	0.0719
	Cost		61.90	Not included	0.0576
Aditi Panda et al.	Time	Cascade Correlation Neural Network	94.76	0.9303	0.1486
SM Satapathy et al.	Time	Stochastic Gradient Boosting	95.24	Not included	0.1632
Ch Prasada Rao et al.	Time	Generalized Regression Neural Network	76.19	Not included	0.0483
	Cost		76.19	Not included	0.0276
Research Project	Time	Neural Networks Ensemble	96.10	0.9920	0.0390
	Cost		91.43	0.9460	0.0857

Table VI: Comparison between different techniques used in literature with current research project.

V. Conclusion and Future Work

In this work a story points approach was used combining an agile effort estimation model and a machine learning model based on the ensemble of deep neural networks. Autoencoders are a technique that allows compressing information and reduces the work of using several techniques like PCA. The encoder block allows large input data dimensions and it reduces information to a small number of features.

When an algorithmic model is combined with machine learning techniques a robust hybrid model ensures reliable results. Cross validation guarantees the generalization capabilities of the algorithm while the ensemble stabilizes the estimates within a range and reduces the standard deviation. In order to improve even more the estimates, data labeling is important so the projects are grouped according to their size which helps the algorithm during training.

From all the techniques used and compared to the original model [8], Adaboost Encoder outperformed in both completion time and total cost since it improved in 16 out of 21 projects and 12 out of 21 projects respectively. The final ensemble did not improve in 5 out of 21 projects for completion time, however the difference between real values and predictions kept under a working week for those 5 projects. In terms of cost, the final ensemble did not improve in 12 out of 21 projects but most predictions kept under 100,000 rupees.

All the techniques proposed did better than the neural networks proposed in literature. Once the hybrid model is proved the next step is adding more machine learning techniques such as Decision Trees, K-Nearest Neighbors (KNN), Random Forest. If there is more information about the projects derived from risk management, more labels can be added to improve the accuracy of predictions.

References

[1] A. Panda, S. M. Satapathy, and S. Rath, "Empirical validation of neural network models for agile software effort estimation based on story points,"vol. 57, 08 2015.

[2] Raúl Rojas, *Neural networks: a systematic introduction*. Berlin: Springer-Verlag, 1996.

[3] C. P. Rao, P. S. Kumar, S. R. Sree, and J. Devi, "An Agile Effort Estimation Based on Story Points Using Machine Learning Techniques,"*Proceedings of the Second International Conference on Computational Intelligence and Informatics Advances in Intelligent Systems and Computing*, pp. 209–219, 2018.

[4] A. T. Raslan et al., "Effort Estimation in Agile Software Proyects using Fuzzy Logic and Story Points,"*50th Annual Conference on statistics, computer sciences, and operation research*, 27-30 2015.

[5] E. Scott and D. Pfahl, "Using developers features to estimate story points," *Proceedings of the 2018 International Conference on Software and System Process*, 2018.

[6] S. M. Satapathy and S. K. Rath, "Empirical assessment of machine learning models for agile software development effort estimation using story points," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 191–200, 2017.

[7] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, 2019.

[8] Z.K. Zia, S.K. Tipu and S.K. Zia, "An Effort Estimation Model for Agile Software Development,"*Advances in Computer Science and its Applications*, vol. 2, no. 1, pp. 314–324, July 2012.

[9] E. Coelho and A. Basu, "Effort Estimation in Agile Software Development using Story Points,"*International Journal of Applied Information Systems,* vol. 3, no. 7, pp. 7–10, 2012.

[10] M. Vyas, et al., "A Review on Software Cost and Effort Estimation Techniques for Agile Development Process," *International Journal of Recent Research Aspects* ISSN: 2349-7688, Vol. 5, Issue 1, March 2018, pp. 1-5.

[11] J. Choudhari and U. Suman, "Story Points Based Effort Estimation Model for Software Maintenance,"*Procedia Technology*, vol. 4, pp. 761-765, 2012.

[12] "The 14th Annual State of Agile Report is Here,"*digital.ai.* https://digital.ai/catalyst-blog/the-14th-annual-state-of-agile-report

[13] R. Popli and N. Chauhan, "Cost and effort estimation in agile software development,"*2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, 2014, pp. 57-61, doi: 10.1109/ICROIT.2014.6798284.

[14] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators,"*Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

[15] M. Fernandez-Diego, E. R. Mendez, F. Gonzalez-Ladron-De-Guevara, S. Abrahao, and E. Insfran, "An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review,"*IEEE Access*, vol. 8, pp. 166768–166800, 2020, doi: 10.1109/access.2020.3021664.

[16] S. Garg and D. Gupta, "PCA based cost estimation model for agile software development projects,"in *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, 2015.

[17] O. Malgonde and K. Chari, "An ensemble-based model for predicting agile software development effort," *Empirical Software Engineering*, vol. 24, no. 2, pp. 1017–1055, 2018.

[18] M. Durán, R. Juárez-Ramírez, S. Jiménez, and C. Tona, "User Story Estimation Based on the Complexity Decomposition Using Bayesian Networks,"*Programming and Computer Software*, vol. 46, no. 8, pp. 569–583, 2020.

[19] M. Gultekin and O. Kalipsiz, "Story Point-Based Effort Estimation Model with Machine Learning Techniques,"*International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 01, pp. 43–66, 2020.

[20] Muhammad Waseem Khan, Imran Qureshi , "Neural Network based Software Effort Estimation: A Survey,"*Int. J. Advanced Networking and Applications,* Volume: 05, Issue: 04, Pages:1990-1995, 2014.

[21] O. Hidmi and B. E. Sakar, "Software Development Effort Estimation Using Ensemble Machine Learning,"*International Journal of Computing, Communication and Instrumentation Engineering*, vol. 4, no. 1, 2017.

[22] Ibtissam Abnane, Mohamed Hosni, Ali Idri, Alain Abran, "Analogy Software Effort Estimation Using Ensemble KNN Imputation", *45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2019

[23] P. S. Kumar, H. Behera, A. K. K, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades,"*Computer Science Review*, vol. 38, p. 100288, 2020.

[24] P. S. Kumar, H. S. Behera, J. Nayak, and B. Naik, "A pragmatic ensemble learning approach for effective software effort estimation,"*Innovations in Systems and Software Engineering*, 2021.

[25] Ziauddin, Khairuz Zaman Khan, Shahid Kamal Tipu, Shahrukh Zia, "An Intelligent Software Effort Estimation System,"*Journal of Expert Systems (JES)*, Vol. 1, No. 4, 2012, ISSN 2169-3064.

[26] H. T. Hoc, V. V. Hai, and H. L. T. K. Nhung, "A Review of the Regression Models Applicable to Software Project Effort Estimation,"*Computational Statistics and Mathematical Modeling Methods in Intelligent Systems Advances in Intelligent Systems and Computing*, pp. 399–407, 2019.

[27] Lujain A. Hussein et al. "Recurrent Neural Network based Prediction of Software Effort,"*International Journal of Computer Applications*, Volume 177 – No.1, November 2017

[28] O. Malgonde and K. Chari, "An ensemble-based model for predicting agile software development effort,"*Empirical Software Engineering*, vol. 24, no. 2, pp. 1017–1055, 2018.

[29] Moocarme, M., Abdolahnejad, M. and Bhagwat, R., 2020. The Deep Learning with Keras Workshop. Birmingham UK: PACKT Publishing.