Effort Estimation in Agile Software Development Using Deep Learning Model

Khalid A. Alsubhi

Assistant Professor

Computer Science Department

Faculty of Computing and Information Technology

King Abdulaziz University, Jeddah, Saudi Arabia

kalsubhi@kau.edu.sa

ABSTRACT

Software effort estimation is a major activity in the process of software development that permits managers and software engineers to accurately estimate the schedule and budget. Although traditional effort estimation approaches are used to estimate effort for agile software projects but they mostly result in inaccurate estimates. According to standard surveys, 30% to 40% of software projects failed because of inaccuracy of software effort predictions. To improve the accuracy and efficiency of effort estimation, an agile software development process is introduced and substitutes the traditional methods in the software industry. One approach of calculating effort of agile projects is the Story Point Approach. Software industries have adapted the Agile model where effort is estimated using Story Points. The goal of this study is to provide a detailed overview of the effort estimation techniques in Agile Software Development based on Story Points. In this paper, we propose a prediction model for estimating story points based on long short-term memory and recurrent network. This model has been applied to a dataset composed of issue reports mined from JIRA repositories. The proposed work uses three different performance metrics i.e. mean magnitude of relative error, mean square error, and prediction to evaluate the performance of the model. The results of the proposed models are compared to the existing models in the literature.

Key words : Effort estimation, story point estimation, deep learning, LSTM.

1. INTRODUCTION

The software in the settings of agile is developed through a number of iterative cycles in short period, facilitating the establishment the change requirements during a project's life where the development software team performs the analysis, designs, implementations, tests processes and then delivers a product increment related to a specific release or version. Each iteration requires the fulfillment of a number of user stories, which are a common communication channel among agile teams to express users' requirements (e.g. sprints in Scrum [22]). Consequently, there is a demand to concentrate on estimating the effort of completing a single user story

rather than the whole project. Obviously, the story points becomes as a most common unit of effort measure for a user story or resolving an issue [24]. In this research, among sixteen open source projects in nine repositories we collected the levels of issue/user story instead of project level as in traditional effort estimation datasets. Our proposed model predict the size of new issues based on the team's previous story point estimates by employing the deep learning architectures using long short-term memory (LSTM). LSTM allows us to model the long-term context in the textual description of an issue. We referred this proposed model as Deep learning model for Agile Software Development Effort Estimation (Deep-ASDEE). This work used as an inspiration for Choetkiertikul et al., (2019) [17]. Deep-ASDEE learns semantic features on behalf of domain experts. The rest of this paper is organized as follows; Section 2 presents a background of the story point effort estimation approach, deep neural networks, and related works in the literature. In Section 3, we then present the Deep-ASDEE model. Section 4 discusses the experimental evaluation of such model before we conclude in Section 5.

2. BACKGROUND

2.1 Story Point Estimation

When a team estimates with story points, it assigns a point value (i.e., story points) to each user story. A story point estimate reflects the relative amount of effort involved in resolving or completing the user story: a user story that is assigned two story points should take twice as much effort as a user story assigned one story point. Many projects have now adopted this story point estimation approach [25]. Projects that use issue tracking systems (e.g., JIRA [28]) record their user stories as issues. An issue is composed of title and description. Projects that use JIRA Agile record story points as well. Story point sizes are used for measuring a team's progress rate, prioritizing user stories, planning and scheduling for future iterations and releases, and even costing and allocating resources as well as for predicting the amount of work delivered, that is in turn quantified in terms of story points from the issues completed within that iteration. Velocity is the sum of the story-point estimates of the resolved issues during an iteration required for planning and predicting when a software version or release should be fulfilled for deployment.

2.2 Related Works

Existing estimation methods are generally classified into three major groups; namely, expert-based, model-based, and hybrid approaches [17]. Expert-based methods depends on human expertise to estimate software efforts. Model-based approaches use the history of previous software projects to predict efforts of new projects. Hybrid approaches combine expert-based methods and model-based [35]. Because of its dynamics, agile projects require alternative approaches for effort estimation [24]. Most existing works focus on estimating effort for a whole project, whilst few works have been presented for building effort estimations models for agile projects [17]. Some recent approaches influenced machine-learning methods to support effort estimation for agile projects. Table 2 illustrates the effort estimation techniques.

3. THE PROPOSED MODEL DEEP-ASDEE

This research is intended to build a prediction system reads an issue title and description as text and generates a story-point estimate for that issue. The title is followed by the description and are combined into a single text document. Documents vector representations are computed, which are used as features to predict the story points of each issue. Figure 2 derived from M. Choetkiertikul et al., [17] shows our proposed Deep learning model for Story point Estimation in Agile software development (Deep-ASDEE).

Figure 1: RNN-LSTM Deep learning model [17]

It is composed of three components (i) document representation using LTSM [], and (ii) the output layer that is adjusted with a simple regression. A title and description of an issue may consists of a sequence of words $s = (w_1, w_2, \dots, w_n)$, e.g., the word sequence (*Standardize XD logging to align with...*) in the title and description of an issue. The input layer is a sequence of words that are embedded and then are fed into the LSTM that in turn outputs a sequence of state vectors, which are pooled to represent a document-level vector, which is then fed into a RNN. Finally, a regressor predicts the outcome.

3.1 LSTM Architecture

Recurrent networks with simple sigmoid cells have unsatisfactory performance for long-sequence dependencies, leading to develop LSTM [1], [2]. LSTM is a special variant of recurrent neural networks [3], designed to learn long-term dependencies through gradient descent. The architecture of LSTMs is composed of units called memory blocks. As illustrated in Figure 2. The memory block is composed of a memory cells, in which their flow of information to hidden neurons are controlled as well as the extracted features are preserved by gates. These gates control flow of information to hidden neurons and preserve extracted features from previous time steps. Each memory cell is a linear unit with self-connected recurrent edge of fixed weight one called “Constant Error Carouse” (CEC), whose activation is considered the state of the cell, ensuring that the gradient can pass across many time steps avoiding vanishing or exploding.

Figure 2: Single LSTM Cell

The memory cell is a vector that stores accumulated information over time; it holds the temporal state of the cell and is capable of maintaining its activation indefinitely. Each memory cell has special logistic units called gates (input, forget, and output and a cell activation) with their own learned weights on connections; that can open or close over time to control flow of information passing through the cell as well as protecting the rest of the network from changing the memory cells contents for multiple time steps [6]. The input gate controls the flow of information into LSTM layer at each time step, the information stored in the memory is refreshed through accepting fresh new input. The forget gate scales the internal state of the cell before adding it as input the cell through the self-recurrent connection of the cell, it learns weights that control the memory stored value decay rate, therefore adaptively forgetting the old cell's memory, irrelevant information. Finally, the output gate controls the information flow out of the LSTM layer [2], [5], [7]. During the training phase, the backward flow of the local error kept fixed within the CEC. Consequently, LSTM tackles this problem and mitigates the effects of gradients vanishing and exploding of RNN, resulting in the network successful to learn long-term sequential dependencies in data. On the contrary, the algorithms employed for training RNNs may limit the memory produced from the recurrent connections, resulting failing to learn long-term sequential dependencies in data. In this section, we illustrate the equations of LSTM hidden layer within a RNN for the (i) activation (forward pass) and (ii) gradient calculation (backward pass). Table 2 illustrates the notations used.

Table 1: The effort estimation techniques in the literature

Method	References	Description	Performance Parameters
COCOMO	B. W. Boehm et al. [27]	construction cost	<i>MRE, MMRE</i>
Presents efforts in different types of software including both traditional and agile development	Z. Khan, et al. [36]	Represent knowledge in terms of If-Then rules.	<i>MRE, MMRE</i>
Cost estimation in Agile software for aerospace Procurement operations	R. Curran, et al. [37]	The integration with a generic methodology to realize the availableprevious knowledge.	R^2
Cost estimation model for estimating cost and effort in agile software projects	S. Kang, et al. [38]	Dynamically Calculates cost for desired features of the software during tracing the project progress	<i>MSE, MMSE</i>
predicting efforts in iterative software development	P. Abrahamsson [40]	Artificial neural networks and regression	<i>RMSE, LOOV – RMSE, MRE</i>
Predicting project velocity in XP projects by utilizing Bayesian Network model	Peter Hearty, et al. [41]	Dynamic Bayesian Network	<i>MRE</i>
Story point agile software estimation using SVR kernel methods	S. Satapathy, et al. [44]	Uses SVR kernel to enhance the accuracy in prediction.	<i>MMRE, PRED</i>
Estimating agile software effort using artificial neural networks based story point technique	A. Panda, et al. [45]	Cascade Correlation NN, General Regression NN, and Polynomial NN.	<i>MMRE, PRED, MSE, R²</i>
Agile cost and effort estimation framework based on particle swarm optimization	I. Manga et al. [46]	Minimizes the estimation accuracy of effort and time.	<i>MRE, MMRE, PRED, TMRE, EMRE</i>
software productivity prediction	P. Sentas, et al. [47]	Multinomial logistic regression	<i>MSE, MMSE</i>
Software productivity and effort prediction	P. Sentas, et al. [48]	ordinal regression	<i>MSE, MMSE</i>
Empirical validation for agile software effort estimation based on story points,	A.Panda, et al. [19]	Neural network models	<i>MSE, MMSE</i>
effort estimation using class points for object oriented systems	S. Kanmani, et al. [49]	Neural network models	<i>MRE, MMRE</i>
Class point based effort estimation of object-oriented systems	S. Kanmani, et al. [50]	Fuzzy subtractive clustering and artificial neural networks	<i>MMRE, PRED, MSE</i>

Software prediction cost with predefined interval estimates	S. Bibi, et al. [51]	Bayesian Belief Networks	<i>MSE, MMSE</i>
Estimating software project effort	M. Shepperd et al., [52]	analogy-based	<i>MMRE, PRED, MSE</i>
Simulation tool for cost estimation	L. Angelis et al., [53]	analogy-based	<i>MSE, MMSE</i>
software effort estimation	F. Sarro et al., [54]	multi-objective evolutionary	<i>MSE, MMSE</i>
Combine the estimates from multiple estimators	E. Kocaguneli [55]		
Estimating story points from issue	S. Porru, et al., [56]	Develop a story point estimation model through extracting TF-IDF features from issue description. The extracted features are fed into SVM classifier or others	<i>MRE, MMRE,</i>
Estimate the effort required to complete an agile project	R. Djouab et al., [57]	Cosmic Function Points (CFP)	<i>MMRE, PRED, MSE</i>
effort prediction model for iterative software development setting	P. Abrahamsson [58]	regression models and neural networks	<i>MMRE, PRED, MSE</i>
Predicting the elapsed time for fixing a bug or the delay risk of resolving an issue	E. Giger [59] L. D. Panjer [60] P. Bhattacharya [61] P. Hooimeijer [62]		
Deep-SE maps text to a numerical score or a class	M. Choetkiertikul et al., [17]	Deep Learning LSTM-Recurrent Highway Network	<i>MMRE, PRED, MSE</i>

Table 2: Notations

Notations	Description
w_{ij}	<i>The weight of the connection from unit i to unit j</i>
a_j^t	<i>The network input to some unit j at time t</i>
b_i^t	<i>The value of the same unit after the activation function has been applied</i>
ι	<i>Input gate</i>
ϕ	<i>Forget gate</i>
ω	<i>Output gate</i>
C	<i>The Set of memory cells of the block</i>
s_c^t	<i>State of cell c at time t (i.e., the activation of the linear cell unit)</i>
f	<i>The activation function of the gates</i>
g	<i>The cell activation functions</i>
h	<i>The output activation functions</i>
I	<i>The number of inputs</i>
K	<i>The number of outputs</i>
H	<i>The number of cells in the hidden layer</i>
O	<i>The objective function used for training</i>

In the conventional RNNs the forward pass is calculated for a length T input sequence x by starting at $t = 1$ and applying the update equations recursively, while incrementing t , and the back-propagation through time backward pass is calculated by starting at $t = T$, and calculating the unit derivatives while decrementing t recursively.

(i) Forward Pass:

a. Input Gates

$$a_i^t = \sum_{l=1}^I \omega_{il} x_i^t + \sum_{h=1}^H \omega_{hl} b_h^{t-1} + \sum_{c=1}^C \omega_{cl} s_c^{t-1} \quad (1)$$

$$b_i^t = f(a_i^t)$$

b. Forget Gates

$$a_\phi^t = \sum_{l=1}^I \omega_{il} x_i^t + \sum_{h=1}^H \omega_{hl} b_h^{t-1} + \sum_{c=1}^C \omega_{cl} s_c^{t-1} \quad (2)$$

$$b_\phi^t = f(a_\phi^t)$$

c. Cells

$$a_c^t = \sum_{l=1}^I \omega_{il} x_i^t + \sum_{h=1}^H \omega_{hl} b_h^{t-1} \quad (3)$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_i^t g(a_c^t)$$

d. Output Gates

$$a_\omega^t = \sum_{l=1}^I \omega_{il} x_i^t + \sum_{h=1}^H \omega_{hl} b_h^{t-1} + \sum_{c=1}^C \omega_{cl} s_c^{t-1} \quad (4)$$

$$b_\omega^t = f(a_\omega^t)$$

e. Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \quad (5)$$

(ii) Backward Pass

$$\epsilon_c^t = \frac{\partial O}{\partial b_c^t} \quad (6)$$

$$\epsilon_s^t = \frac{\partial O}{\partial s_c^t} \quad (7)$$

a. Cell Outputs

$$\delta_i^t = f'(a_i^t) \sum_{c=1}^C g(a_c^t) \epsilon_s^t \quad (8)$$

b. Output Gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^C s_c^{t-1} \epsilon_s^t \quad (9)$$

c. States

$$\delta_c^t = b_i^t g'(a_c^t) \epsilon_s^t \quad (10)$$

d. Cells

$$\epsilon_s^t = b_\omega^t h'(s_c^t) \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + \omega_{cl} \delta_i^{t+1} + \omega_{c\phi} \delta_\phi^{t+1} + \omega_{c\omega} \delta_\omega^{t+1} \quad (11)$$

e. Forget Gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^C h(s_c^t) \epsilon_c^t \quad (12)$$

f. Input Gates

$$\epsilon_s^t = \sum_{k=1}^K \omega_{ck} \delta_k^t + \sum_{h=1}^H \omega_{ch} \delta_h^{t+1} \quad (13)$$

3.2 Document Representation

The second layer is a document representation layer, it may be referred as word embedding layer. We employ the LSTM architecture, a special type of RNN for modeling an issue document sentences by accumulating information sequence, to overcome some of the pitfalls of standard RNNs [1] such as the difficulty to train for long sequences [8]. Word embedding is intended to capture syntactic and semantic information of words. Words are mapped into an n -dimensional “embedding” space $\mathbb{R}^m$ using a look-up matrix of dimensions $v \times m$ where v is the size of the vocabulary and m is the dimension of the embedding space which is the size of each LSTM hidden layer. The output of the document representation layer is fed into an RNN layer for deep representation.

3.3 Pooling

Output vectors are aggregated into a single vector forming the complete document. There are several methods to perform pooling [14]. In this research, we employ a the mean pooling method, which is considered as an effective pooling method [15]

$$h = \frac{1}{T} \sum_{t=1}^T h_t \quad (14)$$

3.3 Regression

The regression process is to produce a story-point prediction by employing the linear activation function in a Forward neural network. This function can be defined as:

$$y = b_0 + \sum_{i=1}^n b_i x_i \quad (15)$$

Where y is the story point prediction, x_i is the input, b_i is the weight, and n is the embedding size.

4. EXPERIMENTAL SETUP

This section presents dataset required for training and testing our proposed model (Deep-ASDEE). The dataset is composed of issue reports mined from JIRA repositories [21] (open source). The dataset is composed of issue types with story points.

4.1 Model Training

In this step, the Document Representation and the parameters of LSTM are initialized for learning. Meanwhile the pre-training, we use the predictiveness of natural language that the next word can be predicted according to previous

words. Using the *softmax* function, at time step t , we can predict the next word w_{t+1} according to the state h_t ,

$$P(w_{t+1} = k | w_{1:t}) = \frac{\exp(U_k h_t)}{\sum_k \exp(U_k h_t)} \quad (16)$$

Where w_t is a word embedding vector, U_k is a free parameter. The probability of the first word $P(w_1)$ in a sequence is the number of sequences, which has that word w_1 starting first. At time step t , h_t is computed by feeding h_{t-1} and w_t to the LSTM unit. In order to measure the quality of a specific set of parameters based on the difference between the predicted and actual story points, the loss function $L(\theta)$ is defined.

4.2 Evaluation Criteria

The accuracy of the proposed model can be evaluated by using Mean Absolute Error (*MAE*), Mean Magnitude of Relative Error (*MMRE*), Prediction at level m (*Pred.*), and the Root Mean Square Error (*RMSE*).

MAE is the average of the absolute values of the prediction errors and is defined in Eq. :

$$MAE = \frac{1}{n} \sum_{i=1}^n |ActualSP_i - EstimatedSP_i| \quad (17)$$

Where, n is the number of issues, *ActualSP_i* is the actual story point, and *EstimatedSP_i* is the estimated story point.

Because of it is more robust for large outliers, we use the Median Absolute Error. *MdAE* is defined as

$$MdAE = Median \{ |ActualSP_i - EstimatedSP_i| \} \quad (18)$$

Where $1 \leq i \leq n$

MMRE is the average of the *MREs*, and it is defined in Eq. .

$$MMRE = \frac{1}{n} \sum_{i=1}^n MRE_i \quad (19)$$

Where *MRE* that measures the deviations between the predicted and the actual values that is defined in Eq.

$$MRE = \frac{|y_i - f(x_i)|}{y_i} \quad (20)$$

Where x_i is the predict actual value and y_i is an observed value.

Pred.(m) measures the percentage of predictions that are within m percent of the actual values.

SA is based on *MAE* and it is defined as:

$$SA = \left(1 - \frac{MAE}{MAE_{guess}} \right) \times 100 \quad (21)$$

Where MAE_{guess} is the MAE of a large number.

5.DISCUSSION

To study the benefits of using LSTM in representing issue reports, similar to the work of M. Choetkietikul et al., [17], we compared the improved accuracy achieved using the features derived from LSTM against that using the features derived from BOW and Doc2vec. Table 3 represents the outcomes. LSTM performs better than BOW and Doc2vec with respect to the *MAE*, *MdAE*, and *SA* measures in twelve projects in ME, UG, and AS projects. Among those twelve projects, LSTM improved over BOW in MS, in DC in terms of *MAE*, in AP, in JI in terms of *MdAE*, and in AP and in TD

in terms of *SA*. It also improved over Doc2vec in MU, in JI in terms of *MAE*, in AS, in JI in terms of *MdAE*, and in TE, in TD in terms of *SA*. In some cases BOW and Doc2vec performs better than LSTM, like MD in terms of *MAE* and *SA*.

Table 3: Evaluation Results of LSTM, BOW, and Doc2vec

Project: ME				Project: JI			
Method	MAE	MdAE	SA	Method	MAE	MdAE	SA
LSTM	1.03	0.86	54.83	LSTM	1.63	1.21	47.34
BOW	1.25	1.28	46.34	BOW	2.00	2.24	36.51
Doc2vec	1.09	0.93	52.65	Doc2vec	2.00	2.04	36.47
Project: UG				Project: MD			
LSTM	1.02	0.81	48.29	LSTM	9.39	9.23	17.01
BOW	1.13	1.22	43.09	BOW	9.71	9.73	14.35
Doc2vec	1.07	0.88	46.16	Doc2vec	7.64	9.40	31.61
Project: AS				Project: DM			
LSTM	1.54	1.33	49.89	LSTM	4.30	3.51	35.91
BOW	1.74	1.46	44.13	BOW	4.55	3.79	32.23
Doc2vec	1.54	1.34	49.89	Doc2vec	4.49	3.80	33.21
Project: AP				Project: MU			
LSTM	2.82	2.67	35.56	LSTM	2.10	2.10	36.89
BOW	2.83	2.70	35.32	BOW	2.20	2.42	34.90
Doc2vec	3.05	2.77	30.75	Doc2vec	2.10	2.56	37.49
Project: TI				Project: MS			
LSTM	2.21	1.88	45.73	LSTM	3.14	2.64	14.57
BOW	2.46	2.19	40.14	BOW	3.15	2.45	14.84
Doc2vec	2.30	2.06	43.83	Doc2vec	3.24	2.79	12.18
Project: CV				Project: TE			
LSTM	2.93	2.64	26.27	LSTM	0.63	0.62	65.25
BOW	2.84	2.79	28.49	BOW	81.90	0.66	56.09
Doc2vec	3.01	2.66	24.48	Doc2vec	0.67	0.85	63.44
Project: DC				Project: XD			
LSTM	0.66	0.59	66.21	LSTM	1.72	1.55	39.04
BOW	0.91	1.06	55.03	BOW	1.89	1.64	33.87
Doc2vec	0.73	0.73	62.99	Doc2vec	1.79	1.65	36.88
Project: BB				Project: TD			
LSTM	0.96	0.95	58.05	LSTM	3.70	4.16	30.61
BOW	1.28	1.20	45.77	BOW	4.28	4.81	20.71
Doc2vec	1.07	1.10	53.82	Doc2vec	4.12	4.57	23.31

To discuss the benefits of using deep representation in representing issue reports, Table 4 illustrates the results of *MAE*, *MdAE*, and *SA* from Deep-ASDEE using LSTM for deep representation of issue reports against the combined LSTM with Automatically Transformed Linear Model,

Support Vector Machine, and Linear Regression Model. The best results are in bold.

Table 4: Evaluation Results of Deep-SE, LSTM+SVM, LSTM+ATLM, and LSTM+LR

Project: ME				Project: JI			
Method	MAE	MdA	SA	Method	MAE	MdA	SA
Deep-ASDE	0.97	0.70	56.99	Deep-ASDE	1.31	1.04	56.69
LSTM+SVM	1.02	0.86	55.26	LSTM+SVM	1.94	1.80	38.14
LSTM+ATLM	1.03	0.90	54.86	LSTM+ATLM	2.00	1.86	36.44
LSTM+LR	1.05	0.91	54.23	LSTM+LR	2.00	1.86	36.44
Project: UG				Project: MD			
Deep-ASDE	0.98	0.76	50.15	Deep-ASDE	5.69	4.70	47.90
LSTM+SVM	1.02	0.99	48.79	LSTM+SVM	6.38	5.18	42.09
LSTM+ATLM	1.01	1.14	33.86	LSTM+ATLM	9.50	9.15	16.11
LSTM+LR	1.33	1.14	33.86	LSTM+LR	9.50	9.15	16.11
Project: AS				Project: DM			
Deep-ASDE	1.30	0.55	57.39	Deep-ASDE	3.59	2.11	45.59
LSTM+SVM	1.39	1.35	54.48	LSTM+SVM	4.00	2.73	39.93
LSTM+ATLM	1.51	1.24	50.75	LSTM+ATLM	4.48	3.56	33.34
LSTM+LR	1.60	1.39	48.36	LSTM+LR	5.05	3.49	25.41
Project: AP				Project: MU			
Deep-ASDE	2.58	2.40	40.55	Deep-ASDE	2.08	1.87	38.18
LSTM+SVM	2.91	2.76	33.58	LSTM+SVM	2.17	2.75	35.66
LSTM+ATLM	2.91	2.63	33.53	LSTM+ATLM	2.34	2.28	30.96
LSTM+LR	3.57	3.49	19.65	LSTM+LR	2.34	2.28	30.96
Project: TI				Project: MS			
Deep-ASDE	1.88	1.28	53.26	Deep-ASDE	3.08	1.90	16.35
LSTM+SVM	1.90	2.00	52.57	LSTM+SVM	3.15	2.94	14.38
LSTM+ATLM	2.39	1.93	41.78	LSTM+ATLM	3.26	2.62	11.63
LSTM+LR	2.58	2.20	37.45	LSTM+LR	3.26	2.62	11.63
Project: DC				Project: XD			
Deep-ASDE	0.65	0.50	66.59	Deep-ASDE	1.55	1.25	45.98
LSTM+SVM	0.71	0.86	63.83	LSTM+SVM	1.71	1.69	37.27
LSTM+ATLM	0.83	0.56	58.64	LSTM+ATLM	1.74	1.57	31.15
LSTM+LR	0.76	0.64	61.87	LSTM+LR	1.76	1.64	32.27
Project: BB				Project: TD			
Deep-ASDE	0.70	0.58	67.85	Deep-ASDE	2.83	2.78	47.34
LSTM+SVM	0.77	0.95	65.29	LSTM+SVM	3.32	3.21	36.51
LSTM+ATLM	1.88	1.70	65.29	LSTM+ATLM	3.68	3.91	36.47
LSTM+LR	1.20	1.10	48.80	LSTM+LR	3.61	3.50	36.47
Project: CV				Project: TE			
Deep-ASDE	2.01	0.76	48.05	Deep-ASDE	0.61	0.56	66.35

LSTM+SVM	2.38	2.21	39.26	LSTM+SVM	0.67	0.86	63.44
LSTM+ATLM	2.96	2.37	25.62	LSTM+ATLM	0.67	0.69	63.34
LSTM+LR	3.20	2.63	20.07	LSTM+LR	0.73	0.68	60.19

6. CONCLUSION

The proposed work has consistently outperformed three common baselines and four alternatives according to our evaluation results. Compared against the *Mean* and *Median* techniques, the proposed approach has improved *MAE* averaging among studied projects. Compared against the *BoW* and *Doc2Vec* techniques, our approach has improved in *MAE*. These are significant results in the literature of effort estimation. A major part of those improvement were brought by our use of the deep learning LSTM architecture to model the textual description of an issue. The use of recurrent networks has improved the predictive performance.

REFERENCES

1. S. Hochreiter and J. Schmidhuber, **Long short-term memory**, Neural Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997. <https://doi.org/10.1162/neco.1997.9.8.1735>
2. F. A. Gers, J. Schmidhuber, and F. Cummins, **Learning to forget: Continual prediction with LSTM**, Neural Comput., vol. 12, no. 10, pp. 2451–2471, 2000.
3. J. F. Kolen, and S. C. Kremer, **Gradient flow in recurrent nets: The difficulty of learning long-term dependencies**, A Field Guide to Dynamical Recurrent Networks, Wiley-IEEE Press, pp. 237–243, 2009.
4. P. Hooimeijer and W. Weimer, **Modeling bug report quality**, in Proc. 22 IEEE/ACM Int. Conf. Automated Softw. Eng., Nov. 2007, pp. 34–44.
5. F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, **Learning precise timing with LSTM recurrent networks**, J. Mach. Learn. Res., vol. 3, pp. 115–143, Mar. 2003.
6. H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, R. Ward. **Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval**, Proceedings of IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), vol. 24, no. 4, pp. 694–707, 2016. <https://doi.org/10.1109/TASLP.2016.2520371>
7. F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, **Learning precise timing with LSTM recurrent networks**, J. Mach. Learn. Res., vol. 3, no., pp. 2451–2471, Aug. 2002.
8. J. F. Kolen, and S. C. Kremer, **Gradient flow in recurrent nets: The difficulty of learning long-term dependencies**, A Field Guide to Dynamical Recurrent Networks, Wiley-IEEE Press, pp. 237–243, 2009.
9. J. L. Elman, Finding structure in time, Cognitive Science, vol. 14, no. 2, pp. 179–211, 1990.
10. A. J. Robinson, **An application of recurrent nets to phone probability estimation**, IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 298–305, August 1994.
11. L. Deng, K. Hassanein, and M. Elmasry, **Analysis of the correlation structure for a neural predictive model with application to speech recognition**, Neural Networks, vol. 7, no. 2, 1994, pp. 331–339, 1994. [https://doi.org/10.1016/0893-6080\(94\)90027-2](https://doi.org/10.1016/0893-6080(94)90027-2)
12. T. Mikolov, M. Karafi'at, L. Burget, J. Cernock'y, and S. Khudanpur, **Recurrent neural network based language model**, in Proc. INTERSPEECH, Makuhari, Japan, September 2010, pp. 1045–1048.
13. A. Graves, **Sequence transduction with recurrent neural networks**, in Representation Learning Workshop, ICML, 2012.
14. T. Pham, T. Tran, D. Phung, and S. Venkatesh, **Faster training of very deep neural networks via p-norm gates**, in Proc. 23rd Int. Conf. Pattern Recognit., 2016, pp. 3542–3547.

15. M. Choetkiertikul, H. K. Dam, T. Tran, T. T. M. Pham, A. Ghose, and T. Menzies, **A deep learning model for estimating story points**, IEEE Transactions on Software Engineering, pp. 1–1, 2019.
16. Alex Graves, **Supervised Sequence Labelling with Recurrent Neural Networks**. <http://www.cs.toronto.edu/~graves/phd.pdf>
17. M. Choetkiertikul, H. Khanh Dam, T. Tran, T. Thi Minh Pham, A. Ghose, and T. Menzies. 2016. **A deep learning model for estimating story points**. IEEE Transactions on Software Engineering, vol. 45, no. 7, July 2019, pp. 637–656.
18. Z. Zia, and S. Tipu SK, **An effort estimation model for agile software development**, Adv Comput Sci Appl vol. 2, no. 1, pp. 314–324, 2012.
19. A. Panda, et al., **Empirical validation of neural network models for agile software effort estimation based on story points**, Procedia Comput Sci, vol. 57, pp. 772–781, 2015.
20. H. Premalatha, C. Srikrishna, **Effort Estimation in Agile Software Development using Evolutionary Cost- Sensitive Deep Belief Network**, the Intelligent Net. and Sys. Society, vol. 12, no. 2, Apr. 2019.
21. Atlassian, **Atlassian JIRA agile software**, 2016. [Online]. Available: <https://www.atlassian.com/software/jira>
22. H. F. Cervone, **Understanding agile project management methods using Scrum**, OCLC Syst. Serv.: Int. Digit. Library Perspectives, vol. 27, no. 1, pp. 18–22, 2011. <https://doi.org/10.1108/10650751111106528>
23. M. Usman, E. Mendes, F. Weidt, and R. Britto, **Effort estimation in agile software development: A systematic literature review**, in Proc. 10th Int. Conf. Predictive Models Softw. Eng., 2014, pp. 82–91.
24. M. Cohn, **Agile Estimating and Planning**. London, U.K.: Pearson Education, 2005.
25. M. Jorgensen, **A review of studies on expert estimation of software development effort**, J. Syst. Softw., vol. 70, no. 1/2, pp. 37–60, 2004.
26. M. Jorgensen and T. M. Gruschke, **The impact of lessons-learned sessions on effort estimation and uncertainty assessments**, IEEE Trans. Softw. Eng., vol. 35, no. 3, pp. 368–383, May/Jun. 2009.
27. B. W. Boehm, R. Madachy, and B. Steece, **Software Cost Estimation with Cocomo II**. Englewood Cliffs, NJ, USA: Prentice Hall, 2000.
28. R. Valerdi, **Convergence of expert opinion via the wideband delphi method: An application in cost estimation models**, INCOSE Int. Symp., vol. 21, no. 1, pp. 1246–1259, 2011.
29. S. Chulani, B. Boehm, and B. Steece, **Bayesian analysis of empirical software engineering cost models**, IEEE Trans. Softw. Eng., vol. 25, no. 4, pp. 573–583, Jul./Aug. 1999.
30. R. Djouab, C. Commeysne, and A. Abran, **Effort estimation with story points and COSMIC function points - an industry case study**, pp. 25–36, 2008. [Online]. Available: <http://cosmicsizing.org/wp-content/uploads/2016/03/Estimation-model-v-Print-Format-adapter.pdf>
31. ISO/IEC JTC 1/SC 7, **INTERNATIONAL STANDARD ISO/IEC Software Engineering COSMIC: A Functional Size Measurement Method**, vol. 2011, 2011. [Online]. Available: <https://www.iso.org/standard/54849.html>
32. P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, **Effort prediction in iterative software development processes—incremental versus global prediction models**, in Proc. 1st Int. Symp. Empirical Softw. Eng. Meas., 2007, pp. 344–353.
33. P. Hearty, N. Fenton, D. Marquez, and M. Neil, **Predicting project velocity in XP using a learning dynamic Bayesian network model**, IEEE Trans. Softw. Eng., vol. 35, no. 1, pp. 124–137, Jan./Feb. 2009.
34. M. Perkusich, H. De Almeida, and A. Perkusich, **A model to detect problems on scrum-based software development projects**, in Proc. ACM Symp. Appl. Comput., 2013, pp. 1037–1042. <https://doi.org/10.1145/2480362.2480560>
35. M. Jørgensen and M. Shepperd, **A systematic review of software development cost estimation studies**, IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 33–53, Jan. 2007.
36. Z. Khairuz Zaman Khan, S. Tipu and S. Zia, **An Intelligent Software Effort Estimation System**, Journal of Expert Systems (JES) 91 Vol. 1, No. 4, 2012, pp. 91–98.
37. R. Curran, P. Watson and S. Cowan, **An Agile Cost Estimating Methodology for Aerospace Procurement Operations**: Genetic CausalCost CENTRE-ing, Aeronautics and Astronautics, book edited by Max Mulder, 2011, pp. 461–486.
38. S. Baik, **Model-based Dynamic Cost Estimation and Tracking Method for Agile Software**, 9th IEEE/ACIS International Conference on Computer and Information Science, 2010, pp. 743–748.
39. M. Cohn, **Bomb Shelter Studies, Agile Estimating and Planning**, Prentice Hall, 2006.
40. P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti and G. Succi, **Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models**, First International Symposium on Empirical Software Engineering and Measurement, IEEE, 2007, pp. 344–353.
41. P. Hearty, N. Fenton, D. Marquez and M. Neil, **Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model**, IEEE Transactions on Software Engineering, Vol. 35, No. 1, 2009, pp. 124–137.
42. AgenaRisk User Manual, Agena, www.agenarisk.com, 2008.
43. S. Satapathy, A. Panda and S. Rath, **Story Point Approach based Agile Software Effort Estimation using Various SVR Kernel Methods**, International Conference on Software Engineering & Knowledge Engineering, 2014, pp. 304–307.
44. S. Satapathy, A. Panda and S. Rath, **Story Point Approach based Agile Software Effort Estimation using Various SVR Kernel Methods**, International Conference on Software Engineering & Knowledge Engineering, 2014, pp. 304–307.
45. Aditi Panda, Shashank Mouli Satapathy and Santanu Kumar Rath, **Neural Network Models for Agile Software Effort Estimation based on Story Points**, Proceedings of the International Conference on Advances in Computing, Control and Networking - ACCN, 2015, pp. 26–30. <https://doi.org/10.15224/978-1-63248-038-5-06>
46. I. Manga and N.V. Blamah, **A particle Swarm Optimization-based Framework for Agile Software Effort Estimation**, The International Journal Of Engineering And Science (IJES), Vol. 3, No. 6, 2014, pp. 30–36.
47. P. Sentas, L. Angelis, and I. Stamelos, **Multinomial logistic regression applied on software productivity prediction**, in Proc. 9th Panhellenic Conf. Informat., 2003, pp. 1–12.
48. P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, **Software productivity and effort prediction with ordinal regression**, Inf. Softw. Technol., vol. 47, no. 1, pp. 17–29, 2005.
49. S. Kanmani, J. Kathiravan, S. S. Kumar, M. Shanmugam, and P. E. College, **Neural network based effort estimation using class points for OO systems**, in Proc. Int. Conf. Comput.: Theory Appl. (ICCTA), 2007, pp. 261–266.
50. S. Kanmani, J. Kathiravan, S. S. Kumar, and M. Shanmugam, **Class point based effort estimation of OO systems using fuzzy subtractive clustering and artificial neural networks**, in Proc. 1st India Softw. Eng. Conf., 2008, pp. 141–142.
51. S. Bibi, I. Stamelos, and L. Angelis, **Software cost prediction with predefined interval estimates**, in Proc. 1st Softw. Meas. Eur. Forum, 2004, pp. 237–246.
52. M. Shepperd and C. Schofield, **Estimating software project effort using analogies**, IEEE Trans. Softw. Eng., vol. 23, no. 11, pp. 736–743, Nov. 1997.
53. L. Angelis and I. Stamelos, **A simulation tool for efficient analogy based cost estimation**, Empirical Softw. Eng., vol. 5, no. 1, pp. 35–68, 2000.
54. F. Sarro, A. Petrozziello, and M. Harman, **Multi-objective software effort estimation**, in Proc. 38th Int. Conf. Softw. Eng., 2016, pp. 619–630.
55. E. Kocaguneli, T. Menzies, and J. W. Keung, **On the value of ensemble effort estimation**, IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1403–1416, Nov./Dec. 2012.
56. S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli, **Estimating story points from issue reports**, Proc. 12th Int. Conf. Predictive Models Data Anal. Softw. Eng., 2016, Art. no. 2.
57. R. Djouab, C. Commeysne, and A. Abran, **Effort estimation with story points and COSMIC function points - an industry case study**, pp. 25–36, 2008. [Online]. Available: <http://cosmicsizing.org/wp-content/uploads/2016/03/Estimation-model-v-Print-Format-adapter.pdf>
58. P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, **Effort prediction in iterative software development processes—incremental versus global prediction models**, in Proc. 1st Int. Symp. Empirical Softw. Eng. Meas., 2007, pp. 344–353.
59. E. Giger, M. Pinzger, and H. Gall, **Predicting the fix time of bugs**, in Proc. 2nd Int. Workshop Recommendation Syst. Softw. Eng., 2010, pp. 52–56.

61. L. D. Panjer, **Predicting eclipse bug lifetimes**, in Proc. 4th Int. Workshop Mining Softw. Repositories, 2007, pp. 29–32.
62. P. Bhattacharya and I. Neamtiu, **Bug-fix time prediction models: Can we do better?** in Proc. 8th Working Conf. Mining Softw. Repositories, 2011, pp. 207–210.
63. P. Hooimeijer and W. Weimer, **Modeling bug report quality**, in Proc. 22 IEEE/ACM Int. Conf. Automated Softw. Eng., Nov. 2007, pp. 34–44.
<https://doi.org/10.1145/1321631.1321639>