Story Point Level Classification by Text Level Graph Neural Network

Hung Phan
hungphd@iastate.edu
Iowa State University
Ames, Iowa, USA

Ali Jannesari
jannesar@iastate.edu
Iowa State University
Ames, Iowa, USA

ABSTRACT

Estimating the software projects' efforts developed by agile methods is important for project managers or technical leads. It provides a summary as a first view of how many hours and developers are required to complete the tasks. There are research works on automatic predicting the software efforts, including Term Frequency - Inverse Document Frequency (TFIDF) as the traditional approach for this problem. Graph Neural Network is a new approach that has been applied in Natural Language Processing for text classification. The advantages of Graph Neural Network are based on the ability to learn information via graph data structure, which has more representations such as the relationships between words compared to approaches of vectorizing sequence of words. In this paper, we show the potential and possible challenges of Graph Neural Network text classification in story point level estimation. By the experiments, we show that the GNN Text Level Classification can achieve as high accuracy as about 80% for story points level classification, which is comparable to the traditional approach. We also analyze the GNN approach and point out several current disadvantages that the GNN approach can improve for this problem or other problems in software engineering.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

graph neural network, story point estimation, term frequency

ACM Reference Format:
Hung Phan and Ali Jannesari. 2022. Story Point Level Classification by Text Level Graph Neural Network. In *The 1st Intl. Workshop on Natural Language-based Software Engineering (NLBSE'22), May 21, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3528588.3528654

1 INTRODUCTION

With the importance of story point effort estimation, there are researches on how to apply the technique to automatically predict the story point of a software task/ issue. To our knowledge, one of the well-known datasets of software effort estimation is the dataset published by Choetkiertikul et al [8]. Machine learning-based approaches have been applied to this dataset for story point estimations by [8]. The approaches range from the classical approach from Bag of Word (BOW) [10] to modern deep learning approaches from Recurrent Highway Network (RHN) and Long Short Term Memory (LSTM). In general, these approaches are formalized as machine learning regression problems.

The input of software effort estimation for each software issue is the title and description of issues. The output is the number of story points that usually ranged from 1 to 100. According to [1], the value of story point should be in the modified version of Fibonacci numbers to ensure that the different story points are not too close to each other. Since then, the distribution of story points is range from 1 to 100 but not in continuous uniform distribution. The input of this problem can be considered as a type of software documentation that is used for requirement collection.

Since Graph Neural Network (GNN) had been applied in several research works in SE [6, 7], it is a new area to apply GNN techniques on software requirement datasets. In this work, we want to study the potential and challenges when adapting a GNN model in text classification for story points categorization. We split the story points label in the dataset [8] into four levels. Next, we build the training model using a technique called Text Level GNN [9]. This training model is used for story points level prediction. We published our code and dataset at this site[1].

2 RELATED WORK

In the problem that we considered as text classification in SE, there are several approaches to applying GNN for this problem. Compared to other deep neural network based model such as in [11] and [12], GNN has an advantages of learning from graphs. The first approach is proposed in [14]. In this work, there are two types of nodes constructed from the graph. They are nodes as documents and nodes as words inside documents. The second approach is called text level graph in NLP [9], which worked by extracting the graph for each input text or document. Compared to [14], [9] is considered as an improvement approach since it can construct a graph for each document that consumes less memory and be feasible to interact with unseen test data. In our work, we inherit the model from [9] for our problem.

3 APPROACH

In this project, we try to solve the story point estimation problem by inheriting the Text Graph Neural Network proposed by [9]. We call our solution as TextLevelGNN-StoryPointEstimator. The overview architecture of our approach can be shown in Figure 1.

[1]https://github.com/pdhung3012/StoryPointEstimation-TextLevelGNN

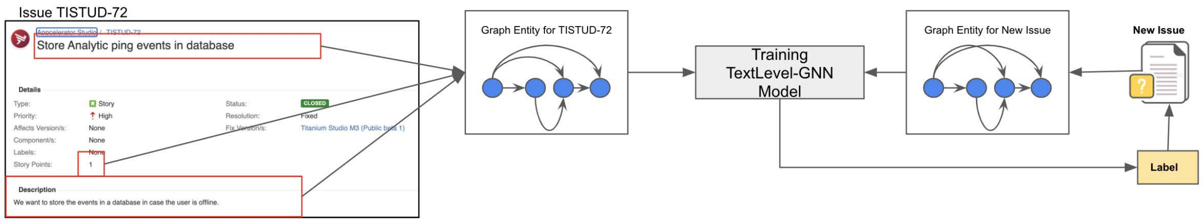
Issue TISTUD-72
Store Analytic ping events in database
Details
Type: Story
Priority: High
Affects Version/s: None
Component/s: None
Labels: None
Story Points: 1
Status: CLOSED
Resolution: Fixed
Fix Version/s: Titanium Studio M3 (Public beta 1)
Description
We want to store the events in a database in case the user is offline.
Graph Entity for TISTUD-72
Training TextLevel-GNN Model
Graph Entity for New Issue
New Issue
Label

Figure 1: Overview Architecture of TextGraph GNN model

The target object given to each issue is the level of story points. According to the distribution of software effort value, we divide the level of story points into 4 levels. They are small (story point is in range [1,5]), medium ([6,15]), large ([16,40]) and huge ([41,100]). This distribution can be summarized in Table 1.The process of issue extraction can be illustrated as follows. First, a text combination of title and description will be concatenated. Second, a graph will be constructed for each issue, which reflects the relationship between each word and its adjacent words. Finally, a GNN model is used to train a model which can infer the software effort levels. Next, we discuss important modules provided in our approach.

3.1 Graph Construction

Each issue contains two textual information: the issue's title and the issue's description. We provide a combination of title and description as the textual representation of a software issue. There are several types of graph construction from the text. For this problem, we inherit the graph construction from [9]. The idea of graph generation is to make an edge from a word to an offset of adjacent words in a sequence. There can be the case that there are edges that appeared multiple times between two specific words. Those cases will be reflected by the edge weight that highlights the popularity of co-occurrence between words. Edges that appeared rarely between words are considered as public edges. The offset of neighbor words between a specific word can be called the sliding window offset w. The sliding window offset is larger, the number of edges is increased, causing the increment of the complexity of the graph.

3.2 Graph Neural Network

One important element to construct a graph neural network is the process of passing information between nodes and edges inside the graph. This mechanism is called Message Passing Mechanism (MPM). In this problem, TextLevelGNN [9] learns the representation for each node by its neighbor and then combines the representations of all nodes/ words to predict the label. The initial vector for each node is retrieved by the vocabulary of Glove vector representation. [3]. The combination mechanism to predict the label can be done by the softmax and Relu functions [9].

4 EVALUATION

4.1 Dataset and Configurations

We use the dataset collected from [8]. There are 16 software projects developed by Agile development, which contains over 23000 software issues and actual story points. Similar to [8], we split the dataset as 80% for training and validation, 20% for testing. We use the same configuration with prior work [9], with batch size as 32, drop out probability as 0.5, sliding window offset as 20, and the default Glove representation pre-trained model with 300-dimensional vectors for each word in English. The distribution of the level of story points and numbers of issues per each level are shown in Table 1. We select the range of story point levels based on the median of story points for each project on dataset [8].

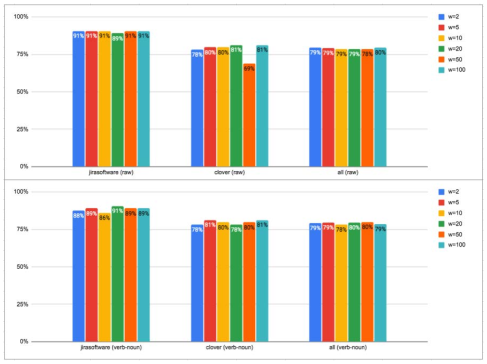

Figure 2: Sensitivity experiment on projects jirasoftware, clover and average accuracy of 16 projects with window size as $[2, 5, 10, 20, 50, 100]$ **and text format as** $[raw, filterverbandnoun]$

We compare the GNN approach with the traditional approach of story point level classification using TFIDF vectorization and Random Forest for classification. For TFIDF vectorization, we use the n-gram as the combination from 1-gram to 4-grams and use the library TfIdfVectorizer in Python [5]. For RandomForest (RF) classification, we use its implementation from Scikit-learn library [4]. We call the traditional approach TFIDF-RF. We aim to answer the following research questions (RQs):

1. **RQ 1**. Accuracy of TextLevel GNN classification from raw text of issues' titles and description.
2. **RQ 2**. Analysis on ability of TextLevel GNN for regression for story points estimation.
3. **RQ 3**. Analysis on abilities of optimization in running time and graph complexity for TextLevel GNN.

Table 1: Distribution of Story Point Level in dataset [8]

Level	SP Value	Size
Small	[1,5]	16759
Medium	[6,15]	5173
Large	[16,40]	1085
Huge	>40	296

Table 2: Accuracy of Text Level GNN compared to. TFIDF-RF in Story Point Level Classification and Story Point Estimation

	Metric	Classify Acc		Reg MAE	
No	Software	TFIDF RF	GNN	TFIDF RFR	GNN
1	appceleratorstudio	78.08%	77.78%	1.66	2.67
2	aptanastudio	46.99%	41.41%	3.50	5.16
3	bamboo	100.00%	100.00%	0.88	0.00
4	clover	83.12%	81.25%	4.04	1.22
5	datamanagement	61.67%	56.36%	6.90	13.40
6	duracloud	99.25%	99.22%	0.91	0.07
7	jirasoftware	91.55%	89.06%	1.84	1.31
8	mesos	97.02%	96.88%	1.30	0.28
9	moodle	59.83%	56.25%	11.17	25.16
10	mule	74.16%	66.41%	2.46	4.03
11	mulestudio	54.42%	52.34%	3.92	5.84
12	springxd	87.96%	88.07%	1.90	1.08
13	talenddataquality	80.14%	83.20%	3.99	1.50
14	talendesb	98.28%	98.44%	0.87	0.14
15	titanium	75.61%	74.55%	2.75	2.30
16	usergrid	95.88%	96.88%	1.25	0.03
	Average	**80.25%**	**78.63%**	**3.08**	**3.09**

4.2 RQ 1. Results on TextLevel GNN Story Point Level Classification

The result of TextLevelGNN is shown in Table 2. From the results on 16 projects, we achieve an accuracy as close as the traditional approach for classification. We achieve the highest accuracy on the bamboo project as 100% while the lowest accuracy is the aptanastudio project as over 41%. We achieve high accuracy as TFIDF and Random Forest in almost all projects. The project that TFIDF and RF outperform us most are mule as about 9% while we achieve significantly higher accuracy on the talendataquality by 3%. We analyze the data inside 16 projects. We see that the mule and talenddataquality provide challenges with low accuracy in story point regression in prior work [8], which our experiment has consistent results in our classification problem.

4.3 RQ 2. From Text Classification to Text Regression

The TextLevelGNN approach [9] is applicable in text or issue classification. Though the space of story point value is not in uniform distribution, the story point estimation should be considered as the text regression problem instead of classification. We study other works in Software Engineering that applied GNN. We see that most of them focus on the classification or code generation instead of regression such as [6]. There are a few problems in SE that can be formalized as regression problems such as project assignment scoring [13] and Github star project prediction [2], however, none of them applied GNN as solutions. We studied several works on graph regression in NLP and see that most of them focus on datasets in different research areas such as image processing [15].

Considering story points as labels for classification A simple solution to convert from classification to regression problem is that we consider the story points themselves as labels for classification. Next, we evaluate the correctness of output by Mean Absolute Error (MAE). We compare the traditional approach TFIDF with Random Forest Regression (TFIDF-RFR). The result which is shown in Table 2, reveals the fact that the TextLevelGNN approach achieves higher MAE than the BOW-RFR approach. It means the TextLevelGNN didn't perform as well as the traditional approach with this simple solution for converting from classification to regression problem. In the upcoming work, we intend to change the mechanism in [9] to regression learning to support story point estimation.

Table 3: Number of edges of projects jirasoftware and clover with window size as $[2, 5, 10, 20, 50, 100]$

window size	jirasoftware	clover
w=2	22511	48658
w=5	58752	124600
w=10	99626	213987
w=20	151170	339562
w=50	229599	568997
w=100	287869	772672

Table 4: Analysis on scale of graphs extracted from training data

Pp	Project	Size	Nodes	Edges	Train Time (sec)
[3]	appceleratorstudio	1868	14969	1218745	244
[3]	aptanastudio	530	8211	545099	41
[3]	bamboo	332	6175	297828	20
[3]	clover	245	5227	339562	14
[3]	**datamanagement**	**2986**	**18490**	**1618297**	**358**
[3]	duracloud	425	3793	289685	21
[3]	jirasoftware	224	2339	151170	9
[3]	**mesos**	**1075**	**25494**	**1698010**	**101**
[3]	moodle	745	8468	696113	70
[3]	mule	568	6133	437088	28
[3]	mulestudio	468	4316	309768	21
[3]	springxd	2256	17198	1194025	128
[3]	talenddataquality	883	7324	445830	92
[3]	**talendesb**	**555**	**8766**	**591293**	**30**
[3]	titanium	1440	23989	1728925	304
[3]	usergrid	308	4114	218491	17
[7]	R8 dataset	4937	2923	1380208	821

4.4 RQ 3. Optimization in Running Time and Graph Complexity

We see that the running time in the training of your approach is longer than the traditional work. The BOW-RF approach requires **4** minutes to complete the training step, while the TextLevelGNN requires about **25** minutes for the same tasks of training on 16 projects of [8]. The testing step achieves almost the same running time for the TFIDF-RF approach and our approach. We study the result and see that one problem due to long training time is the number of edges in our graph is usually high, which can be more than 500000 edges in complicated projects such as *memos*. In future work, we can optimize the running time by simplifying the graph in which we highlight important information.

4.4.1 Analysis on the scale of graph. To study how the complexity can affect the performance of the training process, we analyze the nodes, the edges, the size (number of issues for training) and compare with similar metrics on the popular natural text dataset $R8$ which was used in the work [9]. We have some observations. First, the size of 16 projects is smaller than the R8 dataset. The number of issues for each project ranges from over 200 to 3000. Second, the number of nodes in the R8 dataset is about 3000, which is lower

Table 5: Accuracy on simplifying the graph in verb-noun format in 16 projects

	Text	RawText	Filter Verb-Noun
No	**Software**	**GNN**	**GNN**
1	appceleratorstudio	77.78%	77.95%
2	aptanastudio	41.41%	54.69%
3	bamboo	100.00%	100.00%
4	clover	81.25%	78.13%
5	datamanagement	56.36%	61.72%
6	duracloud	99.22%	99.22%
7	jirasoftware	89.06%	90.63%
8	mesos	96.88%	96.88%
9	moodle	56.25%	61.98%
10	mule	66.41%	67.97%
11	mulestudio	52.34%	50.78%
12	springxd	88.07%	88.07%
13	talenddataquality	83.20%	75.78%
14	talendesb	98.44%	98.44%
15	titanium	74.55%	75.45%
16	usergrid	96.88%	96.88%
	Average	**78.63%**	**79.66%**

Table 6: Accuracy on simplifying the graph in verb-noun format in datamanagement project

Metric	Raw issue	Filter Verb-Noun
nodes	18490	**2986**
edges	1618297	**10309**
training time	358	**209**
accuracy	56.36%	**61.72%**

than some projects like *datamanagement*, *memos*, or *talendesb*. It means that there are more new words inside each issue instead of a standard NLP text dataset. We study some issues and we see that the description of issues is usually longer than the description that the combination of title and descriptions can be more than 10 sentences. This fact of causing the risk of out-of-vocabulary words that didn't appear in the standard vector corpus Glove used in [9]. We can improve this challenge by building a new Glove representation trained from the context of issues' description. Third, the number of edges is much larger than the number of nodes. This is because the mechanism of making edges between a word and neighbor words in the offset $w = 20$ causes the exponential of the edges. Similarly, the number of edges provided in the $R8$ dataset is high which is about 1.3 million edges. In the issues dataset, the project that has the highest number of edges is *memos* project, due to the complexity of long issues' description. The project that has the least number of edges is the $jirasoftware$ project, which can be due to the compact number of training records. Forth, for the running time, the running time for each project ranges from about 9 seconds to 370 seconds, which is comparable with the $R8$ dataset.

4.4.2 Simplifying the graph of issues by verb and noun. We analyze the potential of simplifying the input text as an issue by filtering its words' roles in part of speech tagging by an experiment that its results are shown in Table 6 and Table 5. In this study, we filter the words that are verbs and nouns and remove all words in other tags of the issue. We run the experiment on the *datamanagement* project, we called this configuration $verbNounFilter$. The results show that the accuracy is **56.36%** on this project, compared to **61.72%** with the normal text without the filter. Moreover, the training time and the graph complexity are alleviated. Though this is a very beginning and heuristic strategy to simplify the textual content, it shows the potential of simplifying the graph that focused on important parts of speech tags. The total average accuracy on 16 projects was raised to **79.66%**.

4.4.3 Sensitivity on window size. We study the impact of window size on the number of edges and the classification accuracy by the sensitivity experiment by changing window size to one of the following sizes: $[2, 5, 10, 20, 50, 100]$. We evaluate projects $jirasoftware$ and $clover$. The impact of window size on the number of edges is shown in Table 3, which shows the high increase of edges when we increase the window size. The accuracy experiment is shown in Figure 2. We can see that too big a window size can cause a decrease in the accuracy, possibly due to many non-useful edges of the constructed graph.

5 CONCLUSION

In this project, we propose a new mechanism of story point level estimation. Instead of converting text sequence to vector representation for features, we apply graph as the data structure for story points level classification. By the experiment, we see that the TextLevelGNN approach achieves comparable accuracy compared to the traditional approach using TFIDF for vectorization combined with Random Forest for machine learning classification. There are rooms for improvement in future works, which is related to adjusting the classification problem to the regression problem and optimizing the running time by filtering important information of the input text for graph construction.

REFERENCES

[1] [n.d.]. Fibonacci Agile Estimation. https://tinyurl.com/3cpjbnhx.
[2] [n.d.]. Github project star prediction. https://tinyurl.com/5djdpsc8.
[3] [n.d.]. GLove Tutorial. https://tinyurl.com/bmb2582v.
[4] [n.d.]. Scikit learn tutorial. https://tinyurl.com/29ebj8v9.
[5] [n.d.]. TF-IDF Vectorizer library. https://tinyurl.com/3a6v45kv.
[6] Miltiadis Allamanis. [n.d.]. Graph Neural Networks on Program Analysis. https://tinyurl.com/r2mu7y5x.
[7] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to Represent Programs with Graphs. *CoRR* abs/1711.00740 (2017). arXiv:1711.00740 http://arxiv.org/abs/1711.00740
[8] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. 2019. A Deep Learning Model for Estimating Story Points. *IEEE Transactions on Software Engineering* 45, 7 (2019), 637–656. https://doi.org/10.1109/TSE.2018.2792473
[9] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. 2019. Text Level Graph Neural Network for Text Classification. *CoRR* abs/1910.02356 (2019). arXiv:1910.02356 http://arxiv.org/abs/1910.02356
[10] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets* (2nd ed.). Cambridge University Press, USA.
[11] Hung Phan and Ali Jannesari. 2020. *Statistical Machine Translation Outperforms Neural Machine Translation in Software Engineering: Why and How.* Association for Computing Machinery, New York, NY, USA, 3–12. https://doi.org/10.1145/3416506.3423576
[12] Hung Phan, Arushi Sharma, and Ali Jannesari. 2021. Generating Context-Aware API Calls from Natural Language Description Using Neural Embeddings and Machine Translation. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 - Workshops, Melbourne, Australia, November 15-19, 2021.* IEEE, 219–226. https://doi.org/10.1109/ASEW52652.2021.00050
[13] Milena Vujošević-Janičić, Mladen Nikolić, Dušan Tošić, and Viktor Kuncak. 2013. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Inf. Softw. Technol.* 55, 6 (jun 2013), 1004–1016. https://doi.org/10.1016/j.infsof.2012.12.005
[14] Liang Yao, Chengsheng Mao, and Yuan Luo. 2018. Graph Convolutional Networks for Text Classification. *CoRR* abs/1809.05679 (2018). arXiv:1809.05679 http://arxiv.org/abs/1809.05679
[15] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2021. Graph Neural Networks: A Review of Methods and Applications. arXiv:1812.08434 [cs.LG]