Software Effort Prediction in Agile Development with Machine Learning & Natural Language Processing

Eliska Kloberdanz
Department of Computer Science
Iowa State University
Ames, IA 50011
eklober@iastate.edu

Hung Phan
Department of Computer Science
Iowa State University
Ames, IA 50011
hungphd@iastate.edu

Jeremy Roghair
Department of Computer Science
Iowa State University
Ames, IA 50011
jroghair@iastate.edu

Abstract

The success of software development projects hinges upon, amongst other factors, on project and time management. One popular method used to aid time management and estimating project timelines for agile software development is the estimation number of story points, which represent the amount of development effort per individual software issues or requests in number of man hours. In this paper, we explore various text vectorization machine learning techniques to predict software development effort measured in number of story points. Our results show that the problem can be formulated as both a classification or a regression problem, and successfully solved using supervised learning. Moreover, several of our regression models achieve better accuracy than prior literature. We also demonstrate that Generative Adversarial Network (GAN) semi-supervised yields significantly better results than normal semi-supervised learning. Finally, we show that deep learning architectures such as convolutional attention recurrent neural networks yield very promising results, but require further hyperparameter tuning.

1 Introduction

1.1 Motivation

Estimating effort needed to develop software based individual tasks has become an increasingly important area of research in recent years. Software teams alike rely on effort estimations as a proxy for approximating the amount of time a developer may spend solving a software task. In practice, estimates are most often provided by experts, senior developers, or subject matter experts (SME). The ability to provide these estimates provides teams with the tools for planning, risk mitigation, and monitoring of projects.

1.2 Challenges

Much of the recent work on software effort estimation has focused on a model based approach which applies machine learning algorithms to labeled software estimation datasets [4]. These datasets are often project specific, composed of descriptions of software tasks and an associated labels, which represent the actual amount of effort taken for a developer to complete it. There are two

primary challenges associated with modelling these datasets. The first challenge pertains to natural language processing - the descriptions of software issues are often incomplete, lacking context, and are filled with information that is often irrelevant or difficult to represent. Moreover, representing and interpreting descriptions of software issues using natural language processing (NLP) is often difficult. The process used for representing text as vectors, which can then be fed into a machine learning model, is called vectrorization. While there are known benefits and trade offs between different vectorization approaches, there is no hard-line rule on which one works best. The second challenge relates to modelling - creating one model across various projects is difficult due to the high variability between descriptions and effort, often resulting in models that are not robust for predicting effort accurately. As a result, most prior research has focused on creating models for project specific effort estimation of user stories including work by Choetkiertiku [4] and [8].

A third challenge is that choosing an appropriate machine learning algorithm for representing these vectorizations or word embeddings can be challenging due to limited availability of quality data. Complex models such as deep learning have recently demonstrated incredible performance for modeling natural language tasks [7], however, using such models effectively is often limiting based on the size of the available datasets. Due to this, more conventional machine learning models are typically more suitable for tasks where larger datasets are not as readily available. Software effort estimation is one such task where datasets are difficult to obtain, often due to most software projects occurring in industry where the work is often proprietary and confidential.

1.3 Contributions

In this work, we focus on solving these challenges of software effort estimation using natural language processing and machine learning modeling. In particular, our contributions include:

- Demonstrated applicability of semi-supervised approaches including generative adverserial networks to the problem of software effort estimation for the first time.
- Outperformed prior work on the problem of software estimation for regression-based machine learning models.
- Analyzed a significant number of NLP vectorization techniques and machine learning model types to determine optimal vectorization/model approach.

The structure of this work is as follows. Section 2 provides background and covers related work for software effort estimation. Section 3 presents our methodology for solving the problem of software estimation. Section 4 outlines our experiments and evaluates six important research questions related to software effort estimation. In section 5 we conclude the paper and discuss the future of this work.

2 Background and Related Work

In this section we briefly discuss the background of software effort prediction and related work.

2.1 Background

Generally, software effort estimation can be described as estimating the length of time required for a developer to achieve the desired functionality specified for a given software task. Software development frameworks often define a software task as one of the most basic units of work by which user requirements are measured. An example of this would be a user story within the agile or kanban frameworks. In particular, agile software development is a framework that uses repetitive cycles called sprints, that range between 1-4 weeks, during which time a clearly defined amount of user stories are designed, implemented, tested and delivered into production.

User stories are defined as short descriptions of the desired functional requirements of a software feature, written from the perspective of an end user of the software system. Frequently, these stories can be for a software bug, new feature or rework of an existing feature. Stories are often written with input from a stakeholder or product owner of the software system which dictates requirements, prioritization and scheduling. Software teams begin sprints by determining user stories to include and estimating the amount of man hours necessary for a developer to complete it during the course of a sprint. To this end, stories are associated with story points to provide an estimated time of completion

and measure the size or effort associated with a given story [5]. Most commonly story points are provided by subject matter experts, but are often inaccurate, leading to poor planning and scheduling [17].

Following an agile framework for a software project inevitably leads to a backlog of user stories, consisting of past, present and future user stories to complete. Usually, the backlog consists of the description and story points associated with each user story as well as priority defined by the product owner. It is this backlog of all user stories created during the course of developing a software project that is used as the dataset or input into a machine learning model for estimating software effort for agile stories.

2.2 Related Work

Prior work on the problem of software effort estimation can be categorized into three major areas: SME-based, model-based or a combination of the two. Group SME-based effort estimation through the use of the planning poker technique has shown to be effective for group planning of agile projects, however, it suffers from large overhead of the availability of experts to efficiently utilize [5]. Conversely, model-based approaches are a hands off approach that use previous project story data in order to effectively make a prediction of the amount of effort for new stories. Our work focuses on applying machine learning models to the problem of software effort estimation for agile and is therefore the focus of this section.

One of the earliest models built for software effort estimation is the COCOMO model introduced by Boehm et al. [2]. COCOMO belongs to a class of parametric models developed for software effort estimation. The COCOMO model splits projects into three categories and estimates effort based on different project factors and features of a category. The approach is limiting because the model weights the factors of a project based on external inputs such as human estimations from a table, often resulting in limitations of generalizing the approach to other projects. A survey of parametric models for software effort estimation showed that they are ineffective for accurately predicting effort accurately [15].

More contemporary research has focused on applying NLP techniques with machine learning models for estimating software effort. Ionescu et al. [8] demonstrated the effectiveness of using word embeddings such as TF-IDF and doc2vec [12] with conventional models including a gaussian naive bayes and support vector regression. They showed that different combinations of these embeddings and models were shown to be effective at reducing the mean magnitude of relative error (MMRE). Similarly, Choetkiertikul et al. [4] introduced a deep learning model for predicting story points associated with agile user stories. The approach uses word2vec [12] word embeddings of story descriptions that are fed into a long-short term memory model (LSTM) to create document level embeddings. These are then fed into a recurrent neural network in order to output a prediction of the amount of story points to allocate to a given story. The approach demonstrated significantly less mean absolute error (MAE) compared to alternative models and vectorization choices.

In this work, we follow the trend of the most recent research in the area of software effort prediction. In particular we work towards applying different NLP vectorizations to agile user story descriptions and other machine learning models in order to identify the most accurate and robust model for predicting story points.

3 Approach

The approach we take in this paper consists of three primary steps: (1) data pre-processing and cleaning, (2) text vectorization, and (3) model training.

3.1 Data Pre-processing

We utilize the IEEE TSE2018 dataset, which contains 16 different software projects each containing software issue title, description, and assigned number of story points. The goal of data-processing is to clean the data and prepare it for vectorization. First, we handle missing data by removing data points that lack either both a title and a description of a software issue, or the associated number of story points. Then we convert all words to lower case to ensure that words are not double counted due

to different cases. The next step is tokenization, which involves splitting up sentences into smaller more manageable parts called tokens. Finally, we also remove all non-alphabetic characters and stopwords, which are commonly used words in the English language such as "a" or "the". These words are very generic; and therefore, don't carry any useful information.

We explore the content of title and description for each story points in different representations along with textual representation. From our knowledge, there are two other representations [16, 3]: the Part Of Speech (POS) tags and the dependency parser. The POS tagger assigned forms of each words in a language, while the dependency parser shows the grammar structure including the relationship between words in natural language. We also embed these types of information. In overall, each title and description will be presented by a sequence of text after cleaning,a sequence of POS and a sequence of dependency parse.

3.2 Text Vectorization

After the data is cleaned and tokenized we encode textual data as vectors using various vectorization methods: term frequency inverse document frequency (TF-IDF), Word2Vec [12], GloVe [14], Doc2Vec [10], and LSTM [1]. The TF-IDF encoding approach considers the frequency of tokens in a document with respect to the rest of the corpus. It assigns more importance to terms that occur very frequently in a given document, but infrequently in other documents, which implies relevance. Word2Vec is one of the most popular techniques to learn word embeddings using shallow neural network, which creates vector representations of words. GloVe, or Global Vectors for Word Representation, also creates word embeddings, but is unsupervised. In contrast to the previously mentioned techniques, Doc2Vec create a numeric representation of a whole document as opposed to individual words, which allows for capturing more context. Finally, we also employ a deep learning architectures long short-term memory (LSTM), which is one of the newest more sophisticated vectorization techniques.

3.3 Model Training

Following vectorization we proceed to splitting the data into training and test sets, and training various machine learning models for software effort prediction. We experimented with both classical and deep learning models; however, it was the simpler models that proved to be more effective. The simpler models we conducted experiments on include: XGBRegressor (XGBR), decision trees (DT), random forests (RF), naive bayes (GNB) , logistic regression (LR), linear discriminant analysis (LDA), Neural Network (MLP), support vector machines (SVC for classification and LSVR for regression), quadratic discriminant analysis (QDA) and many others. Our results section presents only the models that yielded the best accuracy. Following model construction and evaluation we also tune model hyperparameters.

Normalize Input and Output Labels for Regression. Although the story points prediction can be considered as regression problem, it is specific compared to general regression problems in Machine Learning. We observe from the data and see that the distribution of story points is imbalance, means there are only groups of story points in a range of story points. For example, in the Moodle project in dataset [4], the story points are in the set of 16 numbers (1-5,8,10,13,20,24,30,40,41,52,60,100). We alleviate the problem by providing a more concrete distribution in labels. We convert the original labels to new scale of label which represents the order of story points instead the actual story points for training. In testing, we provide a module for convert back from ordered label to exact story points. For example, the scale label will be the set from (1-16). We will apply this normalization in some systems with imbalance labels distribution and discuss the result in the Evaluation.

4 Experiments and Results

We formulate six research questions and conduct various experiments to answer them.

4.1 RQ1. Classification: What is the accuracy of story points categories classification?

Prior literature has focused on predicting the exact number of story points for a given software issue description; and therefore, formulated the problem as a regression. RQ1 examines the prediction

accuracy when the problem is formulated as a classification. We split the number of story points into 4 different categories: (1) small: 2 and fewer points, (2) medium: 3 - 8 points, (3) large: 8 - 14 points, and (4) very large: 15 points and more. We show that the accuracy on testing data is close to 70%. Therefore, we conclude that software effort prediction can be formulated and solved as a classification problem.

Table 1: RQ1. Results

Software Project	**Best Accuarcy/ML Model of Vectorizing Techniques**									
	TF-IDF		Word2Vec		Glove		Doc2Vec		LSTM	
	ML Model	Acc %	ML Model	Acc %	ML Model	Acc %	ML Model	Acc %	ML Model	Acc %
mesos	QDA	66.37	LR	64.4	MLP	61.96	RF	60.54	SVC	57.8
titanium	RF	71.52	RF	71.21	RF	70.64	MLP	69.97	SVC	70.46
talendesb	MLP	73.85	SVC	69.35	GBo	71.43	RF	75.23	SVC	69.47
appceleratorstudio	GBo	85.95	SVC	85.51	RF	85.51	RF	85.27	RF	85.78
mulestudio	RF	69.67	MLP	69.13	LR	68.72	MLP	68.58	QDA	68.44
duracloud	SVC	77.48	LR	76.43	RF	74.17	LR	74.17	MLP	75.38
jirasoftware	SVC	74.15	RF	72.44	GBo	73.58	RF	67.9	RF	71.02
clover	QDA	61.72	SVC	55.47	GBo	58.33	RF	54.69	QDA	53.39
talenddataquality	MLP	62.42	MLP	61.89	MLP	60.97	MLP	60.25	RF	59.23
moodle	SVC	54.03	SVC	51.8	GBo	48.8	MLP	48.46	LR	45.03
datamanagement	GBo	54.02	GBo	52.09	GBo	50.07	GBo	47.59	GBo	46.78
springxd	SVC	64.32	QDA	63.84	LR	62.28	GBo	61	LR	57.83
usergrid	LR	71.99	RF	72.41	GBo	72.2	LR	69.92	RF	69.71
bamboo	QDA	69.29	LR	70.25	MLP	68.52	LR	69.1	MLP	68.71
aptanastudio	RF	62.97	LR	62.61	MLP	61.52	MLP	63.57	LR	62
mule	SVC	70.3	LR	69.97	SVC	69.4	MLP	69.4	LR	69.29
Average Accuracy	-	68.13	-	66.80	-	66.13	-	65.35	-	64.40

4.2 RQ2. Regression: What is the accuracy of story points prediction?

RQ2 examines the accuracy of story points prediction formulated as a regression and measured using mean absolute error (MAE). We compare our results to Choetkiertikul et al. (2016) who used the same datasets, but different models and only one vectorization technique. Our results outperform the results of Choetkiertikul et al. (2016) on four datasets and are highlighted in blue in Table 2, which shows the MAE for different models and different vectorization technique and compares them to the MAE published in Choetkiertikul et al. (2016) and shown in the first column of the table. Please not that a smaller MAE indicates a smaller error; and hence, a better result.

We further study the result on the Moodle system. This system has the lowest accuracy in classfication and also has highest MAE compared to other systems in experiments of [4]. Moodle has an imbalance distribution of data as we discussed in Section 3.3. We check the result of TF-IDF with or without labels' normalization. Interestingly, without labels normalization, the best MAE of our approach using TF-IDF is 7.08, while it reduces to 5.55 when we apply the normalization, which outperforms the prior work using LD-RNN. This fact shows the potential of reducing the MAE on

Table 2: RQ2. Results

	Best Mean Absolute Error/ML Model of Vectorizing Techniques										
	LD-RNN [2]	TF-IDF(4 grams)		Word2Vec		Glove		Doc2Vec		LSTM	
Software Project	MAE	ML Model	MAE	ML Model	MAE	ML Model	MAE	ML Model	MAE	ML Model	MAE
mule	2.18	LSVR	2.459	LSVR	2.500	XGBR	2.517	LSVR	2.402	LSVR	2.463
talenddataquality	**2.97**	**XGBR**	**2.768**	**XGBR**	**2.816**	**XGBR**	**2.757**	**XGBR**	**2.807**	**XGBR**	**2.941**
springxd	**1.63**	**LSVR**	**1.617**	**XGBR**	**1.657**	**LSVR**	**1.641**	**XGBR**	**1.663**	**XGBR**	**1.733**
mulestudio	3.23	LSVR	3.588	ABR	3.575	RFR	3.706	MLPR	3.867	LSVR	3.660
duracloud	0.68	XGBR	0.760	LSVR	0.712	XGBR	0.762	XGBR	0.751	LSVR	0.773
mesos	1.02	LSVR	1.122	LSVR	1.157	LSVR	1.197	LSVR	1.199	XGBR	1.189
clover	2.11	XGBR	3.591	LSVR	3.681	LSVR	3.699	XGBR	3.757	LSVR	3.704
datamanagement	3.77	LSVR	5.978	LSVR	5.907	LSVR	6.107	LSVR	6.066	LSVR	6.243
usergrid	1.03	GBR	1.178	LSVR	1.193	LSVR	1.208	RFR	1.125	LSVR	1.184
jirasoftware	1.38	XGBR	1.647	XGBR	1.671	LSVR	1.636	LSVR	1.660	LSVR	1.879
titanium	1.97	LSVR	2.109	LSVR	2.248	LSVR	2.320	LSVR	2.172	LSVR	2.049
appceleratorstudio	**1.36**	**LSVR**	**1.353**	**LSVR**	**1.381**	**LSVR**	**1.308**	**LSVR**	**1.456**	**LSVR**	**1.312**
moodle	**5.97**	**XGBR**	**5.550**	**LSVR**	**7.416**	**LSVR**	**7.802**	**LSVR**	**8.192**	**LSVR**	**6.761**
talendesb	0.64	LSVR	0.861	XGBR	0.878	XGBR	0.855	XGBR	0.897	LSVR	0.943
aptanastudio	2.71	XGBR	3.428	XGBR	3.387	XGBR	3.448	XGBR	3.342	XGBR	3.363
bamboo	0.74	LSVR	0.795	XGBR	0.797	XGBR	0.807	LSVR	0.744	LSVR	0.789
Average MAE	2.09		2.43		2.56		2.61		2.63		2.56

4.3 RQ3. Semi-supervised training: How does the semi-supervised training affect accuracy?

RQ3 investigates whether semi-supervised learning is a viable option for predicting software effort. While the other five research questions are tested on individual software projects datasets separately,

we attempt to answer RQ3 utilizing datasets across various software projects. We use six different software projects as the training data, where one project provides the labeled data and the other five are unlabeled. We use one dataset, which represents a software project different from the ones used during training. To implement GAN for story points prediction, we adopt the approach for movie reviews classification by Li et al ([11]) . We also compare the GAN based approach with the original semi supervised learning published by Dai et al ([6]) .The results are promising. The testing accuracy for GAN semi-supervised learning is higher than both the vanilla semi-supervised approach and the supervised learning approach.

Table 3: RQ3. Datasets

Data	**Software Project**	**Number of Data Points**
Labeled Training Data	appceleratorstudio	3441
Unlabeled Training Data	bamboo, clover, datamanagement, diracloud, jirasoftware	6068
Testing Data	aptanastudio	829

Table 4: RQ3. Results

ML strategy	**Testing Accuracy in %**
Supervised learning	50.6
Normal semi-supervised learning	32.25
GAN semi-supervised learning	52.42

4.4 RQ4. Hyperparameter Tuning: Does hyperparameter tuning improve the accuracy?

RQ4 studies whether hyperparameter tuning improves accuracy on testing data. We select two models: XGB Regressor and a SVM and test if varying the values of various hyperparameters improves the prediction accuracy. In case of XGB Regressor shown in Table 5 the accuracy remains virtually unchanged and in case of SVM shown in Table 6 the accuracy improves, but only slightly. Hyperparameter tuning can be difficult and time consuming, because it requires trying out many different values and combinations of hyperparameter changes. Perhaps, more rigorous hyperparameter tuning would yield a more significant change in testing accuracy.

Table 5: RQ4. Results - Xgb Regressor

Hyperparameter	**Range**	**Best Value**	**Best MAE**	**Default MAE**
objective	linear regression	linear regression	0.8742	0.8785
colsample bytree	0.3	0.3		
learning rate	[0.1,1]	0.1		
max depth	[1,3,5]	5		
alpha	[10,20]	10		
n estimators	[10,100]	10		

Table 6: RQ4. Results - Support Vector Machine

Hyperparameter	**Range**	**Best Value**	**Best Acc**	**Default Acc**
C	[0.1,1, 10, 100]	10	0.7028	0.6935
gamma	[1,0.1,0.01,0.001]	1		
kernel	['rbf', 'poly', 'sigmoid']	rbf		

4.5 RQ5. Vectorization: What text vectorization techniques are most suitable?

We have used various vectorization techniques in our experiments to find the most suitable one. Table 7 shows the prediction accuracy for each vectorization technique averaged across 16 different predictions corresponding to the 16 software projects in our dataset. These results indicate that term frequency–inverse document frequency (TF-IDF) 4 grams performed the best. Our explanation for this result is that TF-IDF reflects the relevance of individual words in a given software issue title and description. The intuition behind this is that if a word occurs multiple times in a issue description, it

is more meaningful than a word that occurs infrequently. Please note that prior to vectorization we remove stop words, which do not provide useful information. Additionally, the 4-gram model takes into account the sequences of words; and therefore, capture context.

Table 7: RQ5. Results - Average prediction accuracy grouped by vectorization method

	Vectorization Technique				
	TF-IDF	Word2Vec	Glove	Doc2Vec	LSTM
Average MAE	2.43	2.56	2.61	2.63	2.56
Average Accuracy in %	68.13	66.80	66.13	65.35	64.40

Table 8: RQ5. Results - Costs and benefits of vectorization methods

Approach	**Pros**	**Cons**
TF-IDF 4 grams	Easy to implement	Cannot run with over 30000 data points and 7-grams
Word2Vec and Glove	Well known approaches, Pre-trained models available	Training vector corpus is expensive Using pre-trained vector will phase out of vocab
Doc2Vec	Solves well with sentence level	Training vector corpus is expensive
LSTM	Newest vectorization technique	Limit in number of words

4.6 RQ6. Are deep learning models suitable for software effort estimation?

RQ6. investigates whether deep learning models are viable for software effort estimation. The results of Choetkiertikul et al. [4] suggested that deep neural networks could be applicable to the problem of software effort estimation. Moreover, they suggested that along with recurrent neural networks, convolutional neural networks are provided with more context and meaning in which to learn than conventional models. Although not the primary focus of our research, we did some preliminary research into determining if either deep learning model type would be viable for the problem of software effort estimation through the use of existing implementations of these models.

To this end, we fit a simple convolutional neural network as suggested by [9], [13] and a recurrent neural network hierachical attention model introduced by Yang et al. [18], [13]. The input data uses a glove word embedding in order to generate a matrix of words from text descriptions. The glove word embeddings are pre-trained on a corpus of words, Wikipedia in our case, and used to generate a vectorized representation of user story descriptions. Each model outputs a categorical prediction of whether a story belongs to a particular class of story points.

Table 9: RQ6. - Accuracy of two existing deep learning approaches using glove word embeddings.

	Model	
Software Project	Simple CNN	Hierarchical Attention
Appceleratorstudio	70.3%	65.7%
aptanastudio	75.5%	69.4%
bamboo	94.9%	92.9%
clover	67.2%	60.3%
datamanagement	55.7%	52.5%
duracloud	95.5%	88.5%
jirasoftware	64.1%	60.3%
mesos	77.9%	71.1%
moodle	33.4%	32.8%
mule	60.3%	60.3%
mulestudio	60.4%	59.5%
Average Accuracy	68.6%	64.9%

Table 9 shows the accuracy of the two deep learning models. Surprisingly, a standard simple CNN model demonstrates consistently better results than the more complex architecture of the hierarchical attention model introduced by Yang et al. These preliminary results demonstrate that a more refined convolutional neural network architecture may provide a more robust model than recurrent neural networks, and another alternative to conventional models shown in table 1.

5 Conclusion and Future Work

In this paper we experiment with various vectorization and machine learning techniques that can be used for predicting software development effort measured in number of story points per each individual issue or request. We utilize the IEEE TSE2018 dataset, which contains 16 different software projects each containing software issue title, description, and assigned number of story points. Our regression results measured in mean absolute error improve upon Choetkiertikul et al. (2016), who used the same dataset. We also show that the problem of predicting number of story points can be formulated as both a classification or a regression problem and successfully solved with supervised learning.In addition to supervised learning solutions we also experiment with semi-supervised learning and show that GAN semi-supervised learning achieves significantly better results than normal semi-supervised learning. Finally, we show that convolutional and attention recurrent neural nets are very suitable for predicting software effort.

Future work could focus on improving the prediction accuracy by further tuning various hyper parameters and exploring other modelling and vectorization techniques for example by building on our deep learning models, and performing further tuning of hyperparameters and model architectures. Another very useful direction would be collecting more training data, because training on a larger datasets could improve the model's generalization ability. Finally, future work could also explore GAN semi-supervised learning or clustering techniques.

References

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[2] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, USA, 1st edition, 2000.

[3] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics.

[4] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya K. Ghose, and Tim Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45:637–656, 2016.

[5] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall PTR, USA, 2005.

[6] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[8] Vlad-Sebastian Ionescu, Horia Demian, and István Gergely Czibula. Natural language processing and machine learning methods for software development effort estimation. *Studies in Informatics and Control*, 26, 2017.

[9] Yoon Kim. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 08 2014.

[10] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[11] Yan Li and Jieping Ye. Learning adversarial networks for semi-supervised text classification via policy gradient. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '18, page 1715–1723, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[13] mlwiz. *NLP Learning Series: Part 1 - Text Preprocessing Methods for Deep Learning*, 2019 (accessed March 23, 2020).

[14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[15] Derya Toka and Oktay Turetken. Accuracy of contemporary parametric software estimation models: A comparative analysis. In *Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '13, page 313–316, USA, 2013. IEEE Computer Society.

[16] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, page 63–70, USA, 2000. Association for Computational Linguistics.

[17] Muhammad Usman, Emilia Mendes, Francila Weidt Neiva, and Ricardo Britto. Effort estimation in agile software development: a systematic literature review. In *PROMISE '14*, 2014.

[18] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. In *HLT-NAACL*, 2016.

Appendix

Contributions

All three authors made high quality important contributions to this project.

Table 10: Summary of Team Member's Contributions

Contribution	Eliska	Hung	Jeremy
Literature Review	1/3	1/3	1/3
Data Pre-processing	1/3	1/3	1/3
Text Vectorization	1/3	1/3	1/3
Machine Learning Models	1/3	1/3	1/3
Report & Presentation	1/3	1/3	1/3

Replication package

This replication package allows for reproducing our results. It includes one link to all 16 raw datasets, and another link to our repository with code, intermediate data, results, test cases, and a readme file. The readme file contains detailed instructions for replicating the results for all six research questions.

Raw Datasets:

`https://github.com/SEAnalytics/datasets/tree/master/storypoint/IEEE TSE2018/dataset`

Repository:

`https://github.com/pdhung3012/SoftwareStoryPointsPrediction`