2024 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA) | 979-8-3503-9463-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/HORA61326.2024.10550870

Enhancing Agile Effort Estimation: An NLP Approach for Software Requirements Analysis

1st Tunahan Catak
Department of Computer Engineering
Kocaeli Univeristy
Kocaeli, Turkey
cataktunahan@gmail.com

2nd Pinar Onay Durdu
Department of Computer Engineering
Kocaeli Univeristy
Kocaeli, Turkey
pinar.onaydurdu@kocaeli.edu.tr

3rd Sevinc Ilhan Omurca
Department of Computer Engineering
Kocaeli Univeristy
Kocaeli, Turkey
silhan@kocaeli.edu.tr

***Abstract*—The accurate estimation of project effort is a crucial objective in software development processes. Given that the communication of task descriptions typically occurs in natural language texts, Natural Language Processing (NLP) methods within machine learning have the potential to provide rapid and effective ways. The objective of this study was to enhance the accuracy of the effort estimation classification task for software requirements by proposing an NLP-based software requirements effort estimation model for agile software development processes. A semi-supervised noise filtering mechanism based on k-means clustering with tf-idf embeddings was implemented and evaluated. The software requirement documents were represented by FastText embeddings, and then a fast text classifier was used to predict the expected effort of a given requirement text. The effectiveness of the implemented model was evaluated and it was revealed that the application of noise filtering has improved the performance with an accuracy of 96.8%.**

***Index Terms*—text classification, word embeddings, fastText classifier, agile effort estimation**

I. Introduction

A software project is a planned effort that involves creating, improving, or maintaining a software system. An accepted project management practice should contain stressing the significance of stakeholder involvement, planning, resource allocation, and quality assurance in reaching effective results [1]. The widespread use of software in almost every aspect of our lives has led to higher expectations from these software products and increased complexity in software development. The competitive nature of the software industry also places significant pressure on developers to deliver results quickly when working on complex software systems. However, it is often observed that the desired success is not always achieved [2], [3].

The reasons for failed software projects [3], [4] can be attributed to several factors, including setting impracticable project objectives, encountering unforeseen levels of project complexity, and making unrealistic effort estimations. Inaccurate effort estimations particularly regarding workforce allocation and scheduling can result in either over- or under-utilization of resources, which in turn can have a detrimental impact on project outcomes.

Effort estimation in software development refers to the process of predicting the expected effort and cost in terms of hours and capital required for a project. It is a critical aspect of project planning since it helps the stakeholders make informed decisions about project timelines, resource allocation, and budgeting. Traditional processes involve detailed analysis of requirements and tasks by using techniques like expert judgment and historical data analysis. This comprehensive approach aims to create a thorough project plan. However, in agile processes, effort estimation focuses on rapid delivery and customer satisfaction. Estimation in agile projects is conducted using ”Story Points” [5] which express the overall effort required to implement a user story or a task, by employing techniques like Planning Poker [6].

Software project effort estimation methods are typically categorized into three main categories in general [7]:

- Algorithmic models: These models are based on mathematical equations and statistical models. The most significant examples include COCOMO [8] and COCOMO-2 models [9].
- Expert estimation-based models: These models rely on project effort estimations provided by experts with domain knowledge in the field of the project. One of the critical issues with these models is the variability in estimation quality depending on the individual's areas of expertise and interest.
- Machine learning-based models: These models apply various machine learning techniques in the scope of analogy-based models which use historical data from similar past projects to predict the duration or cost of a new project [10]. The literature reveals the application of numerous machine learning and artificial neural network models [11], [12], [13], with artificial neural network-based models often cited as the most successful compared to others [14]. Recent studies [7] show that analogy-based models are becoming increasingly important. In addition, analogy-based models, by definition, involve prediction by comparison, and represent a form of expert-based estimation.

Analogy-based models are considered successful compared to other models in the following ways. First, they address both knowledge discovery and knowledge extraction. Second, they only need to deal with problems that arise in practice, whereas algorithmic models must address all potential problems. Third they can handle failure cases, which is valuable as it allows users to identify potentially high-risk situations [15].

In the context of software projects, the communication of task descriptions typically occurs in natural language texts. Thus, Natural Language Processing (NLP) methods within machine learning have begun to be employed for analogy-based models. NLP methods provide rapid and effective ways of identifying similarities and patterns within these textual descriptions.

However, it is challenging to conduct effort estimations by NLP methods due to the informal nature of textual requirements. This informality is due to the presence of diverse domain-specific information, including natural language text mixed with source code and numerical data. It is often the case that different words used in similar contexts should be considered to be similar (polysemy), whereas identical words used in different contexts should not be treated in the same way (ambiguity). The field of NLP provides word embedding methods designed to capture the semantics of the words in specific contexts [16]. Consequently, the selection of word embeddings becomes crucial for the successful implementation of NLP tasks.

Word embeddings are n-sized vector representations of a text in a context. The aim of having different word embeddings is to represent text with a better understanding of the context [17]. There are three main categories of word embeddings which can be listed as conventional, distributed, and contextual methods. Conventional methods like BoW (bag of words) and tf-idf (term frequency-inverse document frequency) are based on term frequency in a document. Although these methods facilitate document representation, they are not sufficient for contextual representation of documents [17]. Distributed models such as Word2Vec, GloVe or fastText have more meaningful insight into a document while representing its content. Contextual models such as GPT, BERT or BART are dynamic models that learn word representation based on the downstream task in the model and according to the input context [17].

The goal of this study is to develop and evaluate an NLP-based software requirements effort estimation model for agile software development processes. The study utilizes a publicly available dataset [18] used for effort estimation tasks in software development. Initially, software requirements data is preprocessed, then its quality is enhanced using a clustering based semi-supervised noise filtering approach. Furthermore, n-skip gram-based word embeddings are implemented using both tf-idf and fastText models. While conventional models struggle with morphological information, distributed models like fastText excel in capturing such details [19]. Therefore, the study evaluates the usability of the proposed clustering based semi-supervised noise filtering mechanism in software requirement effort estimation tasks to enhance data quality. Practitioners and researchers in the software engineering and NLP domain can benefit from the findings of the study since the study provides insights into improving the accuracy and efficiency of effort estimations.

II. Proposed Approach

The study aims to enhance the accuracy of the effort estimation classification task for software requirements. Therefore a semi-supervised noise filtering mechanism based on clustering with tf-idf embeddings is proposed and implemented. After these steps, the software requirement documents were represented by FastText embeddings, and then a fasttext classifier was used to predict the expected effort of a given requirement text.

The general process flow of the proposed model is shown in "Fig. 1". The initial step is data gathering which involves collecting data that contains software project requirements or user story descriptions and their effort estimations. In this study, a subset of the dataset published by Montgomery et. al [18] has been used due to its large collection of software project requirements and up-to-date nature. After having the dataset, preprocessing steps have been applied to have a clear and useful dataset for NLP tasks. In the next step, expert-based labeling was applied for the classification task. Even after preprocessing steps aimed at having clean and useful data, the dataset has been identified with substantial noise. Thus, a semi-supervised clustering technique was proposed and applied as a noise-filtering technique to enhance the quality of the dataset. As a next step, FastText word embeddings are generated for each requirement text. Afterward, a classifier layer has been applied to the training and test set. Finally, a performance evaluation of the model was conducted. Each step will be detailed in subsequent sections.

A. Dataset and Preprocessing

According to the proposed model and the nature of the software effort estimation problem, labeled software tasks were required. Software tasks can be described as user stories created by clients or product owners to define what is requested from the development team. Therefore a large public user story dataset published by Montgomery et. al in 2022 [18] was used in this study since it is one of the recent datasets and contains both user story explanations and effort estimations of 16 public Jira repositories including 1822 projects and 2.7 million issues.

The requirement texts for a software project are typically provided informally and lack a structural format. So, preprocessing steps are important to apply to raw data to make the data more suitable for analysis. This helps to reduce noise, ensure consistency, and improve accuracy in classification tasks. Therefore the following preprocessing steps were implemented. We remove the special characters other than ASCII characters, the user stories that are written in languages other than English. The stop words, long blanks and punctuations were removed. The duplicated user stories were filtered out. We also removed the user stories which don't have any time estimation or 0 estimated.

Moreover, the dataset contains user stories from different projects written by different teams, this reduces the uniformity of the dataset. Therefore, it was decided to use one of the repositories based on the criteria of label distribution of the

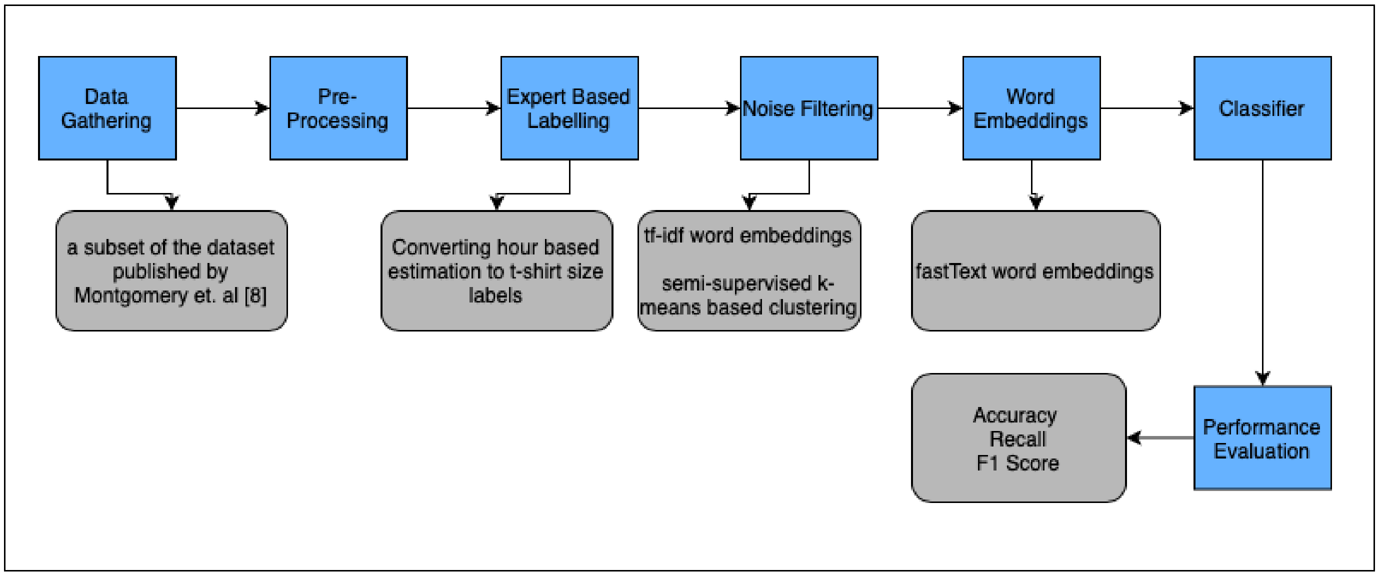


Fig. 1. General process flow of the proposed model.

data and the size of the remaining useful data after the preprocessing steps. As a result, Apache repository dataset, which includes 41155 user stories and effort estimations, was selected.

B. Expert-Based Labeling

Three major software effort estimation metrics are used in software projects nowadays. These are time-based estimations, T-shirt size estimation, and story point estimation [20]. The time-based estimation is the most fragile metric among these three because each team member's velocity in a team is not equal. Software project teams are switching to either story point estimation or t-shirt size estimation [20].

The T-shirt size estimation metric uses generic t-shirt size measures of XS, S, M, L, and XL in agile estimation. This facilitates relative and rapid estimation and project initiation [20].

The unit of the effort estimation is hour-based in the selected dataset. Since using t-shirt size estimation is more efficient for software teams, an expert-based labeling process on the data was applied to convert the amount of time to t-shirt size estimation based on the conversion given in Table I. This conversion algorithm was created by discussing experienced software engineers, project managers, and scrum masters as in [21] and it was determined that 1 day (working day) equals 8 hours.

TABLE I
T-SHIRT SIZE CONVERSION TABLE

Time Based Estimation	T-shirt Size Estimation
t ¡= 1 day	Extra Small (XS)
1 day ¡ t ¡= 2 days	Small (S)
2 days ¡ t ¡= 5 days	Medium (M)
5 days ¡ t ¡= 10 days	Large (L)
10 days ¡ t	Extra Large (XL)

After converting the time estimation column from the required hourly time to T-shirt size, the downstream task turned into a multi-class classification problem with five labels transitioning from the regression problem of time-based estimation. The distribution of the data across various labels following pre-processing and expert-based labeling is shown in "Fig. 2".

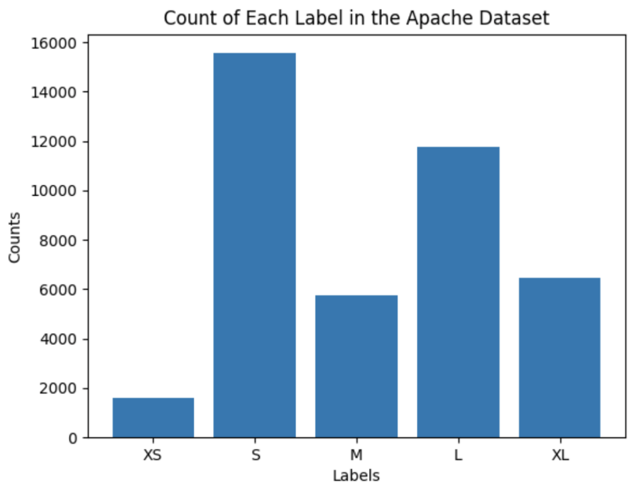


Fig. 2. The distribution of the data in the selected dataset.

C. Noise Filtering

An algorithmic method has been chosen to label data as mentioned in the previous section, since the dataset is composed of more than 40,000 software requirements and it was impractical for experts to label this much data manually. Although this process would create some misclassification in the dataset due to full data that has not been processed by experts, the noise compromises the integrity of the dataset and reduces the accuracy rate.

As Schelling et al. [22] showed the K-means clustering method can be used for noise filtering detection in datasets and its best advantage is simplicity. K-means algorithm is an unsupervised machine learning algorithm that groups related objects in one cluster. Subsequently, the noise data points with a certain radius from the center of clusters can be labeled. Thus, in the study, K-means clustering was selected due to its simplicity and ability to yield favorable outcomes, characterized by high classifier accuracy rates and enhancements in dataset quality.

The high level architecture of the proposed clustering based semi-supervised noise filtering mechanism is shown in "Fig. 3".

The proposed clustering based semi-supervised noise filtering approach starts with creating tf-idf word embeddings of the dataset. tf-idf word embeddings was chosen to be implemented to represent requirements text to be used in K-means clustering since it is simple and fast and complies with the data. After the word embeddings were created for each requirement sentence, the data was grouped into 5 different clusters with K-means-based clustering. The resulting data clusters can be seen in "Fig. 4". The PCA feature reduction method was applied to be able to show the data in 2-dimension.

Afterward, a mathematical formulation was utilized to identify the cluster that best represents the T-shirt size label. This involves tallying the occurrence of each label in every cluster determining the major label for each, and marking the data as a noise when the expert-based estimation is different from the assigned clustering label.

After the data marked with noise and clear tags by the clustering based mechanism, the data which was marked as noise was not immediately discarded. The noise tagged data was applied to the fastText classifier which was trained with the clean tagged data. During the validation, the correctly predicted data marked as clear and appended into the clear dataset. The clustered dataset after the noise filtering was applied can be seen in "Fig. 5"

After the clustering based semi-supervised noise filtering was applied, the dataset contains 27,000 software requirements.

D. *Effort Prediction using fastText Classifier*

The software requirements contain a lot of software engineering-specific words, shortcuts for these words, and team/project-specific meaningful words. Therefore, to find better word embeddings for words that are not in the dictionary, the use of n-grams to represent words would be more useful. In a corpus such as software requirements documents, where character sequences such as technical terms and abbreviations are more meaningful, representing text with character n-grams produces more accurate results. The fastText is an open source library created by Facebook Artificial Intelligence Research (FAIR) for effective learning of word embeddings and classification tasks works with n-grams for better understanding of morphological information of words [23]. FastText provides embedding to the character n-grams. In general, deep neural networks have good performance on natural language processing tasks but beyond that can lead to slower learning, while FastText is faster because it uses a linear classifier to train the model. The FastText classifier [24] has a hidden layer that averages the word representations and a hierarchical softmax layer which is applied to speed up the computation. The model architecture of the FastText classifier is shown in "Fig. 6".

TABLE II
FASTTEXT CLASSIFIER HYPER PARAMETERS [24]

Parameters	Explanation	Selected Value
lr	learning rate	0.05
lrUpdateRate	change the rate of updates for the learning rate	100
dim	size of word vectors	401
ws	size of the context window	5
neg	number of negatives sampled	5
loss	loss function	hs (hierarchical softmax)
thread	number of threads	12

In terms of speed and accuracy, as discussed in the literature, The performance of the fasttext classifier is competitive against many deep learning-based models. Thus in our study, to perform the effort prediction we used FastText classifier [24]. Table II shows the important hyper parameters which are provided by the FastText framework.

In the context of the study, the selected hyper parameter values shown in Table II to have the best validation accuracy for our classification task such as learning rate was set to 0.05, which helps to not overfitting the model; learning rate update rate was chosen to 100 to dynamically adapt the learning rate during training; the dimensionality of the word vectors was selected to be 401, providing a high-resolution feature space to capture semantic relationships. A context window size of 5 and 5 negative samples were chosen to efficiently model word context and negative sampling, respectively; hierarchical softmax (hs) was chosen as the loss function to speed up computation in multiclass classification. Additionally, the model utilized 12 threads to leverage computational resources effectively, thereby speeding up the training process. The validation precision rate was used as the selection criteria of the best hyper parameter value. Dataset was splitted into two parts as training and test set and only the training set was used during the FastText classifier trained with the selected hyper parameters and then the test set was used for calculating the validation precision. Selection of the hyper parameters in each epoch was done by the framework for 4 hours. The framework allows to run the model with the same input document with different hyperparameters.

E. *Evaluation Metrics*

The model performance was evaluated based on the well known evaluation metrics which are precision, recall and F1-Score [25], [26]. Their definitions are given in (1), (2) and (3).

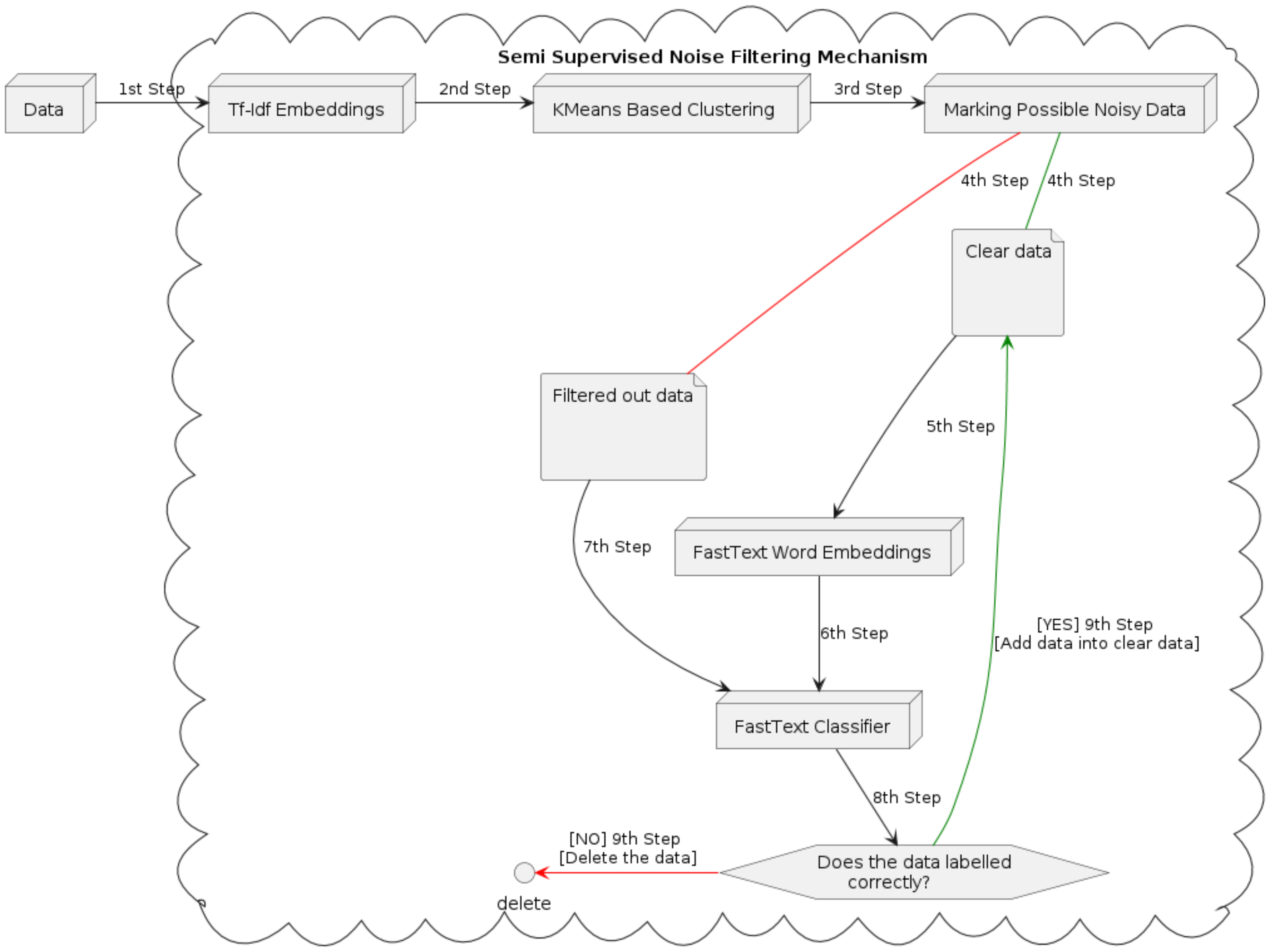


Fig. 3. High level architecture of the proposed clustering based semi-supervised noise filtering mechanism.

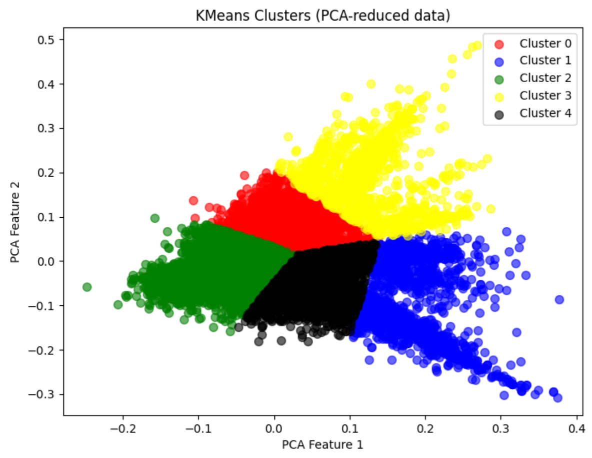


Fig. 4. K-Means clusters the dataset before noise filtering.

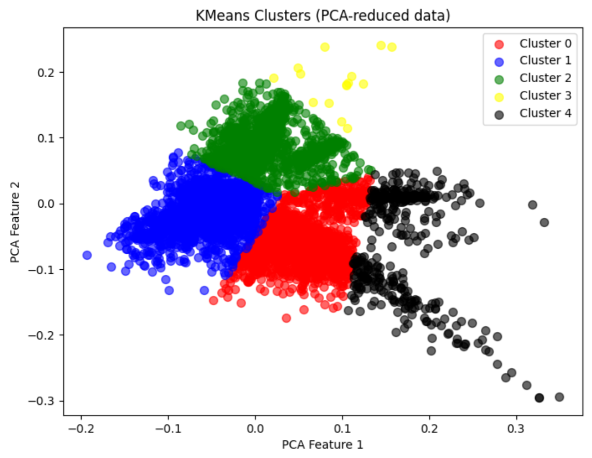


Fig. 5. K-Means clusters the dataset after noise filtering.

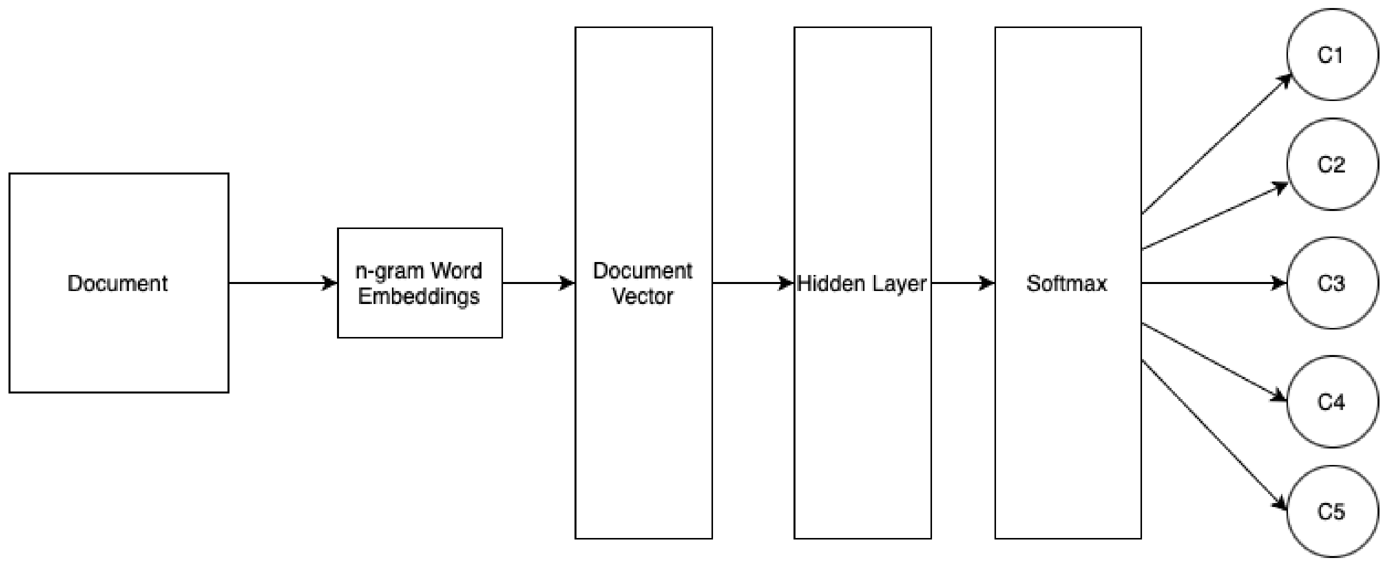


Fig. 6. Model architecture of fastText classifier.

Precision score is a metric which indicates proportion of how many of the values predicted as positive are actually positive (True Positive) and it can be calculated with the following formula:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (1)$$

Recall score indicates how many of the positive class is correctly predicted and it can be calculated with the following formula. Recall is also called as the sensitivity of the model:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (2)$$

F1 score can be calculated with the harmonic mean of Precision and Recall scores.

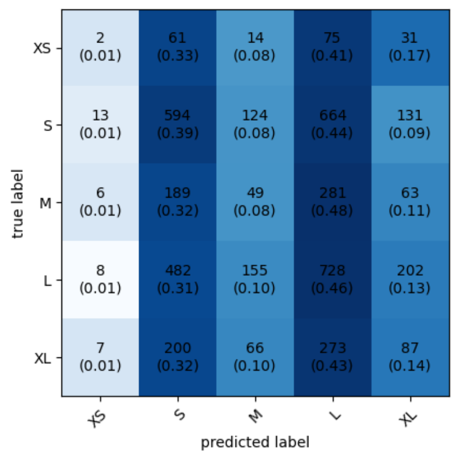
$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3)$$

III. Experiments and Results

This chapter explains two major experiments done to analyze the effect of the proposed clustering based semi-supervised noise filtering mechanism in software project effort estimation domain.

In the first experiment (Ex. ”software effort estimation without noise filtering”) fast text word embeddings were created for the labeled data without applying noise filtering mechanism. After the word embeddings were created, the dataset was splitted two groups (training and test) with 1:5 ratio. The training group was used to train the fastText classifier and fine tuning the hyperparameters. When the training accuracy rate reached the optimum level, the validation set was used to validate the classification task result.

Precision result of the first experiment is around 32% and the confusion matrix as seen in “Fig. 7”. The results indicate



	XS	S	M	L	XL
XS	2 (0.01)	61 (0.33)	14 (0.08)	75 (0.41)	31 (0.17)
S	13 (0.01)	594 (0.39)	124 (0.08)	664 (0.44)	131 (0.09)
M	6 (0.01)	189 (0.32)	49 (0.08)	281 (0.48)	63 (0.11)
L	8 (0.01)	482 (0.31)	155 (0.10)	728 (0.46)	202 (0.13)
XL	7 (0.01)	200 (0.32)	66 (0.10)	273 (0.43)	87 (0.14)

Fig. 7. Confusion matrix of software effort estimation classification task without noise filtering.

that the model without having noise filtering often assigned the software requirements to the most dominant labels like Small and Large. This suggests that the data quality was not good for distinguishing the labels which was possibly due to how the labels were assigned by the experts. Therefore a clustering based semi-supervised filtering mechanism was proposed to enhance the data quality and have a better separation between classes.

In the second experiment (Ex. ”software effort estimation with noise filtering”) fast text word embeddings were created for the labeled data after the proposed clustering based semi-supervised noise filtering mechanism were applied and the clean data has been labeled. After the word embeddings were created, the dataset was splitted two groups (training and validation) with 1:5 ratio. The training group was used to train

TABLE III
EVALUATION RESULTS FOR THE EXPERIMENTS

	software effort estimation without noise filtering	software effort estimation with noise filtering
Precision	32,4%	96,8%
Recall	32,5%	92.9%
F1 Score	33,9%	96,8%

the fastText classifier and fine tuning the hyperparameters. When the training accuracy rate reached the optimum level, the validation set was used to validate the classification task result.

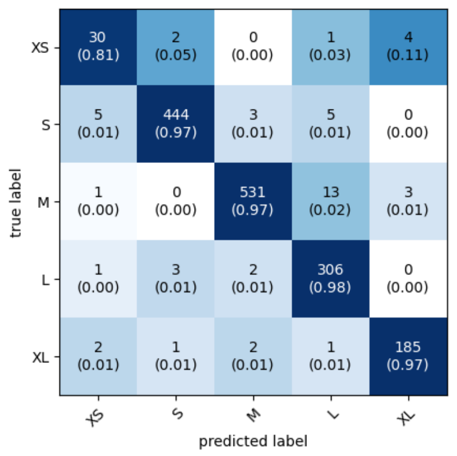


Fig. 8. Confusion matrix of software effort estimation classification task with noise filtering.

Precision result of the second experiment (”software effort estimation with noise filtering”) is around 96% and the confusion matrix as seen in “Fig. 8”. The results clearly show the proposed noise filtering mechanism is enhancing data quality in our dataset and significantly improves accuracy rate of the classification task.

Precision was not the only evaluation metric that was used in the study to understand the benefits of the proposed noise-filtering mechanism. Besides the precision, recall and f1 scores are calculated and evaluated. The Table III shows the comparison of these metrics regarding the two experiments.

When noise filtering was not applied, the precision, recall, and F1 score were significantly lower, at 32.4%, 32.5%, and 33.9% respectively. Conversely, the application of noise filtering dramatically improved performance across all metrics: precision soared to 96.8%, recall increased to 92.9%, and the F1 score matched the precision at 96.8%. These results highlight the critical role of noise filtering in enhancing the accuracy and reliability on datasets of software effort estimations.

IV. CONCLUSION

The aim of the study was to develop and evaluate a NLP based software requirements effort estimation model for agile software development processes. In order to achieve this a clustering based semi-supervised noise filtering mechanism was proposed and its effectiveness was evaluated by applying it to an already prepared software requirement dataset [18] by conducting two different experiments.

In the first experiment, the dataset was used in a fastText classifier model without the proposed noise filtering mechanism being applied. And in the second experiment, the same dataset was used in a fastText classifier model with the proposed noise filtering mechanism being applied. The results show that using K-Means clustering noise filtering mechanism is valuable to have great accuracy in NLP based effort estimation for the software engineering domain.

As part of the future work, exploring to apply other state of art clustering mechanisms for noise filtering, using only unsupervised methods and comparing methods efficiency in this domain were planned. In addition, to measure the versatility of the proposed noise filtering model, the clustering based semi-supervised approach will be applied into other domains for classification tasks.

REFERENCES

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. Palgrave Macmillan, 2005, google-Books-ID: bL7QZHtWvaUC.

[2] R. N. Charette, “We waste billions of dollars each year on entirely preventable mistakes.”

[3] N. Neha, A. Tandon, G. Kaur, and A. G. Aggarwal, “Synergic impact of development cost and slippage cost on software delivery time,” *International Journal of System Assurance Engineering and Management*, Jan. 2023. [Online]. Available: https://doi.org/10.1007/s13198-022-01850-8

[4] A. Idri, I. Abnane, and A. Abran, “Missing data techniques in analogy-based software development effort estimation,” *Journal of Systems and Software*, vol. 117, pp. 595–611, Jul. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121216300334

[5] M. Cohn and R. C. Martin, *Agile estimating and planning*, ser. Robert C. Martin series. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2006, oCLC: ocm59138872.

[6] S. Bilgaiyan, S. Sagnika, S. Mishra, and M. Das, “A systematic review on software cost estimation in agile software development,” *JOURNAL OF ENGINEERING SCIENCE AND TECHNOLOGY REVIEW*, vol. 10, pp. 51–64, 08 2017.

[7] N. Halimawan, F. W. P. Dharma, W. , and N. Surantha, “Model and Dataset Trend in Software Project Effort Estimation – A Systematic Literature Review,” 2021. [Online]. Available: https://doi.org/10.24507/icicel.15.10.1109

[8] B. W. Boehm, “Software Engineering Economics,” *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4–21, Jan. 1984. [Online]. Available: http://ieeexplore.ieee.org/document/5010193/

[9] P. Musilek, W. Pedrycz, Nan Sun, and G. Succi, “On the sensitivity of COCOMO II software cost estimation model,” in *Proceedings Eighth IEEE Symposium on Software Metrics*. Ottawa, Ont., Canada: IEEE Comput. Soc, 2002, pp. 13–20. [Online]. Available: http://ieeexplore.ieee.org/document/1011321/

[10] V. Resmi, S. Vijayalakshmi, and R. S. Chandrabose, “An effective software project effort estimation system using optimal firefly algorithm,” *Cluster Computing*, vol. 22, no. S5, pp. 11 329–11 338, Sep. 2019. [Online]. Available: http://link.springer.com/10.1007/s10586-017-1388-0

[11] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira, “A GA-based feature selection and parameters optimization for support vector regression applied to software effort estimation,” in *Proceedings of the 2008 ACM symposium on Applied computing*. Fortaleza, Ceara Brazil: ACM, Mar. 2008, pp. 1788–1792. [Online]. Available: https://dl.acm.org/doi/10.1145/1363686.1364116

[12] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the Value of Ensemble Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, Nov. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6081882/

[13] P. Suresh Kumar, H. Behera, A. K. K, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades," *Computer Science Review*, vol. 38, p. 100288, Nov. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1574013720303889

[14] I. F. De Barcelos Tronto, J. D. S. Da Silva, and N. Sant'Anna, "Comparison of Artificial Neural Network and Regression Models in Software Effort Estimation," in *2007 International Joint Conference on Neural Networks*. Orlando, FL, USA: IEEE, Aug. 2007, pp. 771–776, iSSN: 1098-7576. [Online]. Available: http://ieeexplore.ieee.org/document/4371055/

[15] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 736–743, Nov. 1997. [Online]. Available: http://ieeexplore.ieee.org/document/637387/

[16] E. M. D. B. Fávero, D. Casanova, and A. R. Pimentel, "SE3M: A Model for Software Effort Estimation Using Pre-trained Embedding Models," Jun. 2020, arXiv:2006.16831 [cs]. [Online]. Available: http://arxiv.org/abs/2006.16831

[17] D. S. Asudani, N. K. Nagwani, and P. Singh, "Impact of word embedding models on text analytics in deep learning environment: a review," *Artificial Intelligence Review*, vol. 56, no. 9, pp. 10 345–10 425, Sep. 2023. [Online]. Available: https://link.springer.com/10.1007/s10462-023-10419-1

[18] L. Montgomery, C. Lüders, and W. Maalej, "An Alternative Issue Tracking Dataset of Public Jira Repositories," in *Proceedings of the 19th International Conference on Mining Software Repositories*, May 2022, pp. 73–77, arXiv:2201.08368 [cs]. [Online]. Available: http://arxiv.org/abs/2201.08368

[19] B. Kuyumcu, C. Aksakalli, and S. Delil, *An automated new approach in fast text classification (fastText): A case study for Turkish text classification without pre-processing*, Jun. 2019, pages: 4.

[20] R. K. Mallidi and M. Sharma, "Study on Agile Story Point Estimation Techniques and Challenges," *International Journal of Computer Applications*, vol. 174, pp. 9–14, Jan. 2021.

[21] I. Dan, R. Catalin, and O. Oliver, "An NLP Approach to Estimating Effort in a Work Environment," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. Split, Croatia: IEEE, Sep. 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9238219/

[22] B. Schelling and C. Plant, "Kmn - removing noise from k-means clustering results," in *Big Data Analytics and Knowledge Discovery*, C. Ordonez and L. Bellatreche, Eds. Cham: Springer International Publishing, 2018, pp. 137–151.

[23] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.zip: Compressing text classification models," Dec. 2016, arXiv:1612.03651 [cs]. [Online]. Available: http://arxiv.org/abs/1612.03651

[24] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," Aug. 2016, arXiv:1607.01759 [cs]. [Online]. Available: http://arxiv.org/abs/1607.01759

[25] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, Jul. 2009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0306457309000259

[26] H. M and S. M.N, "A Review on Evaluation Metrics for Data Classification Evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, pp. 01–11, Mar. 2015. [Online]. Available: http://www.aircconline.com/ijdkp/V5N2/5215ijdkp01.pdf