International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Automated Requirement Prioritisation Technique Using an Updated Adam Optimisation Algorithm

Pratvina Talele*[1], Rashmi Phalnikar[2]

Submitted: 28/04/2023 **Revised**: 28/06/2023 **Accepted**: 06/07/2023

Abstract: Requirement Engineering plays a crucial role in developing software and focuses on gathering requirements from stakeholders with diverse interests. The optimisation algorithm aims to select features to give meaningful information about requirements. These features can be used to train a model for prioritising requirements, which remains a challenging and complex task. The study aims to understand which optimisation algorithms consider suitable features and assign priority to the requirement. The study shows that the existing Adam Algorithm needs to be more capable of assigning accurate priority to the requirement due to the sparse matrix generated for the text dataset and being computationally costly. It was also unable to consider the requirements dependency. This paper suggests an improved approach to prioritise requirements called the Automated Requirement Prioritisation Technique (ARPT) to overcome the limitations of the Adam Algorithm. Compared to Adam Algorithm, the ARPT method results show a wide gap of mean squared error of 1.29 against 6.36. The execution time of the proposed method is 1.99ms as against the Adam algorithm, which is 3380ms. Based on our work described in the paper, it can be concluded that the results of ARPT in assigning priorities to requirements have improved by 80% compared to the Adam algorithm.

Keywords: *Machine Learning, Natural Language Programming, Optimisation Algorithm, Requirement Engineering, Requirement Prioritisation*

1. Introduction

Requirement engineering (RE) covers identifying customer requirements, analysing them, figuring out feasibility, looking for a workable solution, clearly defining the goals and a solution, validating the documents, and managing requirements as they are turned into a functional system. Requirement engineering is the methodical use of tried-and-true concepts, processes, tools and notation to define a proposed system's expected behaviour and constraints [1]. Elicitation, specification, validation, and management of requirements tasks must be completed during this phase [2]. Different software versions can be issued in succession due to constraints like budget and time during the development phase. Therefore, deciding which requirements should be considered in the software's initial release is crucial. The requirement prioritisation (RP) procedure is where stakeholders assign priorities to the requirements [3]. Stakeholders decide the priority of requirements to be implemented in the succession of software releases.

Requirements prioritisation has proven incredibly difficult, with scalability being one of the most significant challenges [4],[5]. In the case of large-scale projects, there are a vast number of stakeholders. There may be conflict about prioritising criteria due to these stakeholders' divergent needs. Meeting stakeholder requirements and raising future expectations in a cost-effective, secure, timely, and relevant way is the main challenge today's companies face. It may be challenging for requirements analysts to choose which requirements should be prioritised due to budget and production schedule restrictions, but this will likely result in high customer satisfaction. Due to a lack of standards, incorrect requirement prioritisation may prevent end users from rejecting the programme, resulting in an unsuccessful product. High-priority requirements must be considered before low-priority requirements to maximise the cost benefits and meet software deadlines. Prioritisation techniques are getting more and more critical. Most research is conducted in this field, yet using the suitable method or framework at the right time might take time and effort. Implementing security features has been recommended since the beginning of software, which is the requirement phase [4].

The Analytic Hierarchy Process (AHP), Must have, Should have, Could have, Won't have (MoScoW), Bubble sort, Binary Search Tree, Hundred Dollar and Priority group are just a few of the requirements prioritisation algorithms utilised by stakeholders [6]. Researchers have recently developed numerous prioritisation methods, including Apriori [7], Gradient Descent Rank [8], Adaptive Fuzzy Hierarchical Cumulative Voting [9], and DRank [10]. The study conducted by Talele and Phalnikar [11] demonstrates that the complexity and scalability of current prioritisation algorithms are constrained.

During the elicitation and specification phases, stakeholders

[1]*School of Computer Engineering and Technology, Dr. Vishwanath Karad MIT World Peace University, Pune, Maharashtra, India, 411038*
[2]*School of Computer Engineering and Technology, Dr. Vishwanath Karad MIT World Peace University, Pune, Maharashtra, India, 411038*
** Corresponding Author Email: pratvina.talele@mitwpu.edu.in*

identify and classify software requirements. This step of the software development cycle is also automated by various machine learning methods [12]. Similarly, this study aims to assign priority to the requirements by considering requirements dependencies more accurately using the optimisation algorithm. The main contribution of the work is twofold. First, an automated method is proposed to assign priority to requirements. Second, to evaluate the proposed method, this study compares the existing optimisation algorithms to prioritise requirements in terms of mean squared error and execution time. The objective of this study is to address the following research questions –

- RQ1: Is the priority generated by ARPT by considering requirement dependencies more accurate than the priority generated by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques?
- RQ2: Is ARPT less time-consuming than Adam?

The datasets generated during the current study are available from the corresponding author upon reasonable request.

The rest of the paper is organised as follows. Related work is discussed in section 2. The automated method and results are described in sections 3 and 4. Section 5 describes the conclusion and future work.

2. Related Work

The importance of requirements prioritisation can be seen from a variety of perspectives. Prioritising requirements is one of the most critical roles for decision-makers [13]. According to Firesmith, requirement prioritising is an essential activity in software engineering since it sets the sequence of requirements to be implemented and delivered to stakeholders in different software versions depending on their relevance [14]. As a result, requirement prioritisation refers to the order in which requirements are prioritised or implemented [6]. GDRank considers user decisions and ordering criteria. To gather requirements, Quality Function Deployment (QFD) is employed. AHP is used for pair sampling in Pattern Driven Architectural Partitioning, which balances functional and non-functional requirements. It provides output based on a decision maker's priorities expressed as a set of Ordered Requirements Pairs, and estimates the final approximated priority of the requirement. Human users' effort to create pair preferences increases significantly as the requirements vary. [8]. In the Apriori method, two functions, join and prune, are performed repeatedly to find frequently occurring items. By combining them, a set of requirements is generated. The pruning procedure looks for requirements in the database. It satisfies the number of stakeholders who support a given criterion whereas ignoring all others [7].

In WhaleRank Optimisation Algorithm, the ranking constants generate a linear rank by combining four ranking functions based on dictionary words, similarity measure, management perception, and updated requirements. WOA calculates the best weights for the requirements to obtain the best priority for the ranking constants [15]. The Grey Wolf Optimisation (GWO) technique is based on grey wolves' natural foraging behaviour. This method's primary goal is to use a population of search agents to identify the best solution to a given problem. Regarding the time required to prioritise requirements and assign a number to each requirement to indicate which is more or less significant than the others, requirement prioritisation GWO performs better than the AHP technique [16]. The Whale Optimization (WO) method is based on the bubble-net technique employed by humpback whales for foraging. The Grey Wolf Optimization (GWO) technique solves optimisation problems by considering grey wolf hunting behaviour. [6] presents a hybrid model for prioritising software requirements that incorporates the strengths of the Whale and Grey Wolf optimization algorithms (WGW).

To minimise the requirement for professional participation during improving the performance, SRPTackle provides a semiautomated approach combining a generated requirement priority with a multi-criteria decision-making technique, clustering techniques, and a binary search tree. SRPTackle's performance was evaluated using seven tests that used a standard dataset from a large real-world software project [17]. Case Based Ranking method is an advanced version of AHP. It accepts a preference elicitation method that mixes sets of preferences gathered from stakeholders with limitations built automatically using ML techniques and leveraging information about (partial) requirements orders. It disregards requirement dependency and the inclusion of additional requirements [18].

To prioritise the requirements, many authors have developed various requirement prioritisation methods. Requirement prioritisation is an agile method of determining the essential requirements for the successful development of software.

- Analytic Hierarchy Process (AHP) – To calculate the priority of one requirement over another, all pairs of requirements are compared.
- Case-Based Rank (CBRank) – AHP version accepts a priority elicitation method that combines stakeholder-defined priorities with priorities obtained using machine learning techniques. It does not always consider requirement dependencies.
- Must have, Should have, Could have, Won't have (MoSCoW) – In MoSCoW, the team distributes the client's requirements into four categories, i.e., Must, Should, Could and Won't. The requirement is then assigned priority, which is a complex process.

As per the study in [11], all these methods work only for a small set of requirements and need the user's involvement to execute these methods.

Due to a lack of specialisation of stakeholders, classification of requirements and assigning priorities to these criteria leads to prejudice [19]. Regarding accuracy, scalability, and complexity, the most often used prioritisation algorithms, such as Cumulative Voting and Analytic Hierarchy Process, perform poorly [11],[20]. The RP process steps can be automated by assigning priority values to requirements and minimising pairwise comparisons among requirements. Deep learning, reinforcement learning, and case-based rank are all machine learning algorithms that can be utilised to solve this challenge. Machine Learning algorithms such as Case based rank, GDRank, and Apriori could be better in complexity and scalability. The limitation of the GDRank algorithm is that it works only for a small set of requirements and is time-consuming. Whale Rank Optimisation algorithm, Grey Wolf algorithm and WGW algorithm are also time-consuming and need expert involvement to perform prioritisation manually. There needs to be more considering the requirement's dependencies. SRPTackle algorithm execution also heavily depends on the experts' participation and does not consider requirements dependencies. Apriori and Case Based Rank algorithms manually work only for a small set of requirements and don't consider requirements dependencies. These approaches need expert involvement and are complex and time-consuming. To overcome these challenges, there is a need to study existing optimisation algorithms to check the accuracy and scalability performance and consider the requirements dependencies.

3. Methodology

3.1. ARPT: Proposed Architecture

Prioritising software requirements is an important aspect of requirement engineering. To prioritise the requirements, many researchers have developed various requirement prioritisation methods. The requirement prioritising process is an iterative method of finding the essential requirements for the successful development of software or systems.

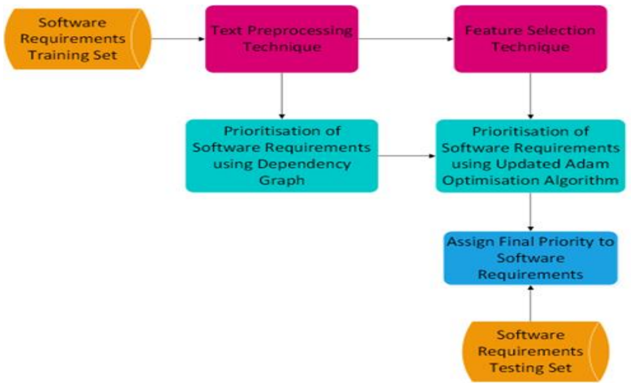
Software Requirements Training Set
Text Preprocessing Technique
Feature Selection Technique
Prioritisation of Software Requirements using Dependency Graph
Prioritisation of Software Requirements using Updated Adam Optimisation Algorithm
Assign Final Priority to Software Requirements
Software Requirements Testing Set

Fig. 1. ARPT: Proposed Architecture

Based on the literature review, different machine learning algorithms are proposed to prioritise requirements, such as CBRank, GDRank. Many areas of research and engineering focus significantly on gradient descent-based optimisation algorithms. Many problems in these areas can be presented as optimising an objective function that requires maximisation or minimisation of its parameters. Adam is an efficient stochastic gradient descent optimisation method that requires first-order gradients and consumes little memory. Adam is a multifaceted algorithm that can handle large-scale machine learning challenges. It works better for sparse datasets [21]. This study considers a large set of requirements mentioned in the natural language. After preprocessing this dataset, massive sparse data are generated. Adam algorithm is well suited for dealing with this sparse data. Existing optimisation algorithms such as stochastic gradient descent (SGD), mini-batch SGD, SGD with Momentum, and Root Mean Squared Propagation (RMSProp) [21],[22] are used to compare with the proposed method that finds the priority for requirement as per the steps shown in Fig. 1. Details of the steps –

1. Data Preparation – This involves two phases, Text Preprocessing and Feature Selection
   a. Text Preprocessing – All requirements are given in plain English. These requirements may include numbers and punctuation marks that may not provide precise information. Text is converted to lower text. Extra white space, punctuation marks, and stop words are eliminated [23].
   b. Feature Selection – The TF-IDF technique provides numerical input to machine learning algorithms and identifies features that can give more information about the requirement.
2. The priority is calculated using a dependency graph to consider the dependency of requirements.
3. The Adam optimisation algorithm is updated to determine the priority for the sparse data in the case of text data and consider the priority calculated using a dependency graph. This Updated Adam algorithm is used and compared to stochastic gradient descent (SGD), mini-batch SGD, SGD with Momentum, and Root Mean Squared Propagation (RMSProp) algorithms.

Algorithm 1

Priority Dependency Graph(Req)

```
1. $Priority\_DG\_FR_i = \sum_{i,j}^{k}(NFR_j \rightarrow FR_i)$
2. $Priority\_DG\_NFR_j = Type\_NFR_j + \sum_{i}^{k} Priority\_DG\_FR_{i\rightarrow k}$
```

Algorithm 2

Updated Adam SGD(X, y)

```
1. Initialise
    a. epochs = 1000
    b. $\eta$ = 0.01
    c. $\beta$ 1 = 0.9
    d. $\beta$ 2 = 0.999
    e. $\varepsilon$ = 10e-8
    f. i = 1
2. do
//Objective Function
    a. y_predicted = f(X, y, w, b)
//Considers the dependency of requirements
    b. y_predicted = Priority DG + y predicted
    c. Find Gradients w grad, b grad using $\delta$i f(X,y predicted)
    d. Update First Moment Vector, FMV
    e. Update Second Moment Vector, SMV
//error divide by zero due to sparse nature of text data – do bias correction in both moment vectors
    f. FMV = FMV / (1 - ($\beta$ 1i) + $\varepsilon$)
    g. SMV = SMV / (1 - ($\beta$ 2i) + $\varepsilon$)
    h. Update Learning Rate, $\eta$
    i. Find Gradients w, b
while i in range(epochs)
3. Return w, b
```

The Adam (Adaptive moment estimation) algorithm utilises the benefits of two optimisation algorithms: AdaGrad [24] and RMSProp [25]. AdaGrad performs better with sparse gradients and RMSProp in non-stationary and online environments. It uses estimations of the first and second moments of the gradients to calculate individual adaptive learning rates for distinct parameters. To achieve the advantages of AdaGrad and RMSProp, Adam changes the first-moment vector and second-moment vector based on the past gradients:

$$\text{AdaGrad\_moment\_vector} = (\beta1^{i}) * \text{AdaGrad\_moment\_vector} + (1 - \beta1^{i}) * \text{grad}^{2} \quad (1)$$

$$\text{RMSProp\_moment\_vector} = (\beta2^{i}) * \text{RMSProp\_moment\_vector} + (1 - \beta2^{i}) * \text{grad}^{2} \quad (2)$$

During initial iterations and values of β1 and β2 are close to 1, FMV and SMV are biased towards zero. The sparse data are enormous because the software requirements are mentioned in terms of text. Therefore, the division factor becomes zero during the Adam algorithm's bias correction step. To overcome this drawback, the ε is added to the division factor. The AdaGrad moment vector and RMSProp moment vector are updated as in (3) and (4), respectively.

$$\text{AdaGrad\_moment\_vector} = (\text{AdaGrad\_moment\_vector})/((1 - (\llbracket \beta1 \rrbracket \text{^}i) + \varepsilon)) \quad (3)$$

$$\text{RMSProp\_moment\_vector} = (\text{RMSProp\_moment\_vector})/((1 - (\llbracket \beta2 \rrbracket \text{^}i) + \varepsilon)) \quad (4)$$

Similarly, the learning rate is updated as in (5).

$$\eta = (\eta * \llbracket (1 - \llbracket \beta2 \rrbracket \text{^}i) \rrbracket \text{^}(1/2))/(((1 - \llbracket \beta1 \rrbracket \text{^}i) + \varepsilon) \quad (5)$$

To improve the efficiency of an algorithm, the gradient (grad) is changed during every iteration, as in (6).

$$grad = grad - (\eta * AdaGrad\_(moment\_vector))/(\llbracket RMSProp\_(moment\_vector) \rrbracket \text{^}(1/2) + \varepsilon) \quad (6)$$

This work validates the hypothesis by applying the proposed method to software projects. The projects should be large-scale projects, well documented and can be evaluated to construct the ground truth. These projects are verified by the Project Coordinator. After preprocessing the data, this ground truth has a requirements list prioritised in the project. The dataset includes 11 projects, 37 functional and 56 non-functional requirements and the requirement's priorities and dependencies. The non-functional requirement types are 'Security', 'Usability', 'Performance', 'Availability', 'Maintainability', 'Qualitative Attribute', 'Testability', 'Portability', 'Support Module' and 'Software Quality Attributes'. The details of the project include the parameters given in Table 1.

Table 1. Project Details

Sr. No.	***Parameters***
1.	Project Title
2.	Functional Requirements 1 – 5
3.	Priority of Functional Requirement 1 – 5 (scale range 1 – 5)
4.	Non-Functional Requirements 1 – 5
5.	Type of Non-Functional Requirements 1 – 5
6.	Priority of Non-Functional Requirement 1 – 5 (scale range 1 – 5)
7.	Are Non-Functional Requirements 1 – 5 dependent on Functional Requirements?
8.	Objectives of the project

The first step of the proposed method, text preprocessing and feature extraction algorithms, are executed on these requirements and 271 features are extracted. To explain the Updated Adam algorithm, consider an example of one requirement – "Mobile battery life should be high". The features extracted from this requirement are battery, life and high. These features are available only in one of the requirements from the whole dataset. Adam algorithm should map these features to functional requirement. But it does not converge as expected. These features should give more information about the type of software requirement. So, the Adam algorithm is updated as during the bias correction of both moment vectors, the "ε" is added to the division factor as shown in Equations 3 and 4. Therefore, the Updated Adam algorithm converges as expected for dense features; values of these vectors will decrease and for the sparse features such as "battery" and "life", values of these vectors will increase.

3.2. Example

The priority of functional requirement 'i' is calculated by considering the dependency of non-functional requirements on functional requirements. i.e., the more non-functional requirements dependent on functional requirement 'i', the more important functional requirement 'i' is. The priority of non-functional requirement is produced based on the type of non-functional requirement and the priority of functional requirements on which it depends e.g., if the type of non-functional requirement is 'Security', it is given the highest priority than that of the 'Support Module'. These calculated priorities are given as input to train the model and predict the new priorities. The new predicted priorities using the proposed model are compared with the ground truth values to check the accuracy.

Project Title - Agile software management

Table 2 shows all the details of the project "Agile software management", i.e., requirements and type of requirements and priorities assigned to requirements.

Where, FR – "Functional Requirement" and NFR – "Non-Functional Requirement"

- As per the proposed methodology, all requirements are pre-processed and features are extracted first.
- To calculate the priority dependency value for functional requirements, the dependency of non-functional requirements on functional requirements is considered and calculated using step 1 of algorithm 1. Using this step, the priority dependency value for functional requirements are produced as – FR1 – 3, FR2 – 3, FR3 – 3, FR4 – 2.
- To calculate the priority dependency value for non-functional requirements, the type of non-functional requirement and the dependency value of functional requirements is considered and calculated using step 2 of algorithm 1. Using this step, the priority dependency value for non-functional requirements are produced as – NFR1 – 7, NFR2 – 7, NFR3 – 8, NFR4 – 8, NFR5 – 5.

Table 2. "Agile software management" Project Details and Sign Conventions used

Requirement	***Type***	***Priority***	***Sign Conventions***
Scrum master is able to create new projects and assign members to it	FR	5	FR1
Team members create sprints and move tasks from sprint backlog to todo, in progress, completed, accepted	FR	5	FR2
Scrum master is able to view progress graphs of each member	FR	4	FR3
Team members collaborate via comment section and task reviews	FR	3	FR4
Page load time less than 2 secs	Performance	5	NFR1
The website is available all the time as long as the user is connected to the internet	Availability	5	NFR2
Easy navigation and user friendly ui	Usability	5	NFR3
All data are sanitized before it is inserted in the database, preventing sql injections	Security	5	NFR4
Testing to be automated via selenium webdriver, test cases can be performed at any time to perform testing	Testability	4	NFR5

These priority dependency values will be used in algorithm 2, "Updated Adam SGD," to calculate the final priority. Table 3 shows the new calculated priority values.

The error between ground truth and priority values obtained using the proposed algorithm is 0.198.

Table 3. "Agile software management" Project Details and new calculated priority values

Requirement	***Type***	***Priority***	***Priority_DG***	***Priority_Upadated Adam***
Scrum master is able to create new projects and assign members to it	FR	5	3	4.48
Team members create sprints and move tasks from sprint backlog to todo, in progress, completed, accepted	FR	5	3	4.82
Scrum master is able to view progress graphs of each member	FR	4	3	4.3
Team members collaborate via comment section and task reviews	FR	3	2	4.48
Page load time less than 2 secs	Performance	5	7	3.9

The website is available all the time as long as the user is connected to the internet	Availability	5	7	4.31
Easy navigation and user friendly ui	Usability	5	8	3.91
All data are sanitized before it is inserted in the database, preventing sql injections	Security	5	8	4.322
Testing to be automated via selenium webdriver, test cases can be performed at any time to perform testing	Testability	4	5	4.69

4. Results

In this part, we elaborate on the specific description and structure of the experiments using the standard rules established by Wohlin et al. [26] for reporting and documenting software engineering experiments.

4.1. Experiment Description

The studies were driven by the need to evaluate the effectiveness of ARPT against that of Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD and GD. The capacity of a prioritising technique to generate speedy and accurate prioritisation outcomes is represented by its effectiveness. As a result, the evaluation is carried out to assess the accuracy of the results as well as the execution time. Similarly, the following is our experiment's research questions and hypotheses.

The objective of requirement prioritisation is to facilitate users in selecting the most important requirements, considering constraints such as stakeholder expectations and requirements interdependences. Therefore, correct prioritising requirements is an important step to ensure a successful software release. Hence, the efficacy of a particular requirement prioritisation approach may be determined by how well it facilitates users in determining a preference aligned with the ground truth. The subject of the effectiveness assessment might be interpreted as the following research question.

- RQ1: Is the priority generated by ARPT by considering requirement dependencies more accurate than the priority generated by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques? This question can be divided into the two sub-questions and hypotheses listed below.
  - RQ1.1: Is ARPT prioritising results less error cost than the prioritisation results provided by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques?
    - H1.1$_{0Error}$: The ARPT and a specific technique (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD) have the same error cost.
    - H11.1$_{1Error}$: The ARPT and a specific technique (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD) do not have the same error cost.
  - RQ1.2: Are the ARPT prioritising results more accurate than the prioritisation results provided by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques?
    - H1.2$_{0Mean\ Square\ Error}$: The ARPT and a specific technique (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD) have the same mean square error.
    - H1.2$_{1Mean\ Square\ Error}$: The ARPT's mean square error and the mean square error of a specific approach (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD) are not the same.

The time required by each technique was calculated.

- RQ2: Does ARPT require less time than Adam?
  - H2$_{0ExecutionTime}$: ARPT's execution time for generating the prioritisation list is similar to that of a specific approach (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD).

- $H2_{1\text{Execution Time}}$: The time ARPT requires to generate the prioritisation list differs from that required by a specific approach. (Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD).

4.2. Experimental Outcomes

The results of the experiments carried out to address the research questions stated in Section 4.1 are analysed. We used statistical analysis to put the presented hypotheses to the test. To implement the prioritisation algorithms, user-friendly Python 3.8.5 is used. It supports a variety of dataset file types, including CSV. Scikit-learn was also selected because it contains several modules that make creating machine learning classifiers and computing the factors used in the evaluations more easily. A DELL Laptop computer was used, with an Intel(R) Core (TM) i5-10300H processor 2.50GHz, 8 GB RAM, running a 64-bit Windows Operating System. The dataset is split into two parts with an 8:2 ratio. 80% dataset is used to train the model and 20% dataset from the same domain is used to test the model.

RQ1: Is the priority generated by ARPT by considering requirement dependencies more accurate than the priority generated by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques? This question can be decomposed into the following two sub-questions and corresponding hypotheses.

RQ1.1: Is ARPT prioritising results less error cost than the prioritisation results provided by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques?

All algorithms, GD, Mini-batch SGD, SGD with momentum, RMSProp and ARPT, are executed for 1000 epochs. It is observed that error cost is reduced in the case of these algorithms, as shown in Fig. 2. The error cost starts from 0.611 in the case of GD when the iteration starts, whereas it is 5.94 in the case of SGD at iteration number 1000. The Adam algorithm forms the distorted waveform, creating substantial sparse data for the text requirements. Therefore, the Adam algorithm takes more time to reduce the error cost. The ARPT algorithm achieves a higher rate of convergence than the Adam algorithm. Thus, the first null hypothesis $H1.1_{0\text{Error}}$ is rejected.

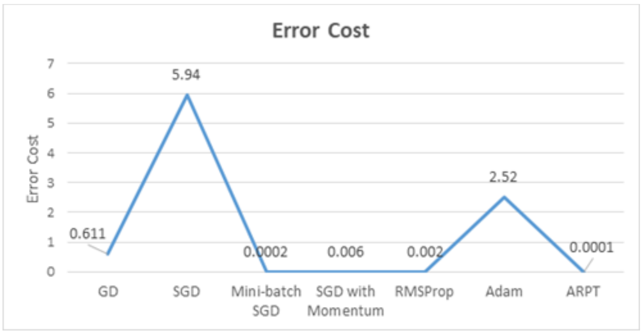

Fig. 2. Comparison of Optimisation Algorithms (Error Cost)

RQ1.2: Are the ARPT prioritising results more accurate than the prioritisation results provided by Adam, RMSProp, SGD with momentum, Mini-batch SGD, SGD, and GD techniques?

The findings from Fig. 3 have shown that the difference between the predicted priority and the original priority, i.e., mean squared error calculated using ARPT, is lesser than that of existing optimisation algorithms. It is also seen that the mean squared error is different for all types of optimisation algorithms, so the null hypothesis $H1.2_{0\text{Mean Square Error}}$ should be rejected. In conclusion, ARPT outperformed Adam by 80%.

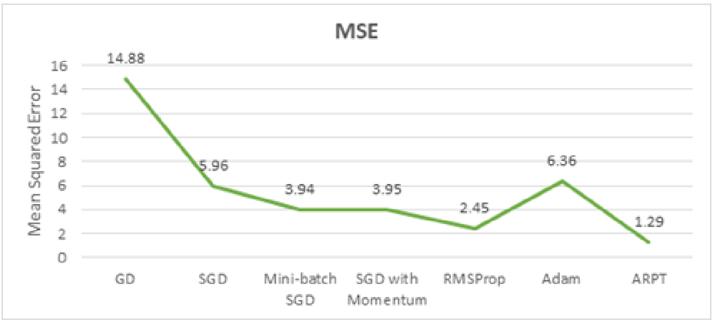

Fig. 3. Comparison of Optimisation Algorithms (Mean Square Error)

RQ2: Is ARPT less time-consuming than Adam?

It is observed that the Adam algorithm takes more time 3380ms to assign priority to requirements than that of other optimisation algorithms. The execution time for the ARPT method is reduced to 1.99 ms, as shown in Fig. 4. The execution time for ARPT differs from that of other optimisation algorithms, so the null hypothesis $H2_{0ExecutionTime}$ is also rejected.

As a result, the statistical studies clearly showed that the null hypothesis is invalidated with reasonable certainty, and the alternative hypotheses $H1.1_{1Error}$, $H1.2_{1Mean\ Square\ Error}$, $H2_{1ExecutionTime}$ are validated. Also, ARPT outperforms GD, SGD, Mini-batch SGD, SGD with momentum, RMSProp and Adam in requirement prioritisation.

The results of this study showed that the requirements dependencies could enhance the efficiency of prioritising requirements. The advantage of ARPT over existing methods is that ARPT considers requirements dependencies to determine the priorities of the requirements. This enhances the accuracy of the prioritisation values. As a result, ARPT is especially helpful for recently established IT organisations entering new domains where experts are difficult to find.

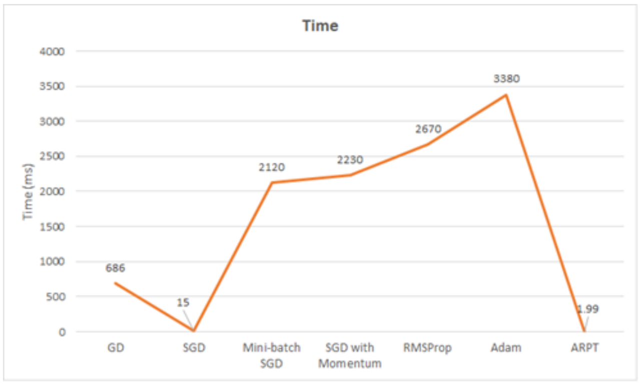

Fig. 4. Comparison of Optimisation Algorithms (Execution Time)

5. Threats to Validity

Experiment-based research is frequently prone to many sorts of validity threats (for example, conclusion, internal, construct, and external validity) [26]. We tried to mitigate and remove these threats in this research, yet, a number of these threats are beyond our control.

- Internal validity – Internal validity threats affect studies without our knowledge and/or are beyond our control. The analysis of computing time for assigning priorities to the requirements for every method constitutes a threat to internal validity due to the measurement's extreme subjectivity about the system's configuration. As a result, all algorithms must be implemented in the same prioritising context. To mitigate this threat, the execution time of ARPT is compared with other optimisation algorithms. The comparison could indicate when to prioritise; nevertheless, implementing language variations in the techniques' resources is an additional threat. Furthermore, whereas the computer system's configuration used to execute the algorithms can influence their performance, the algorithms compared herein were executed on a system with the configuration mentioned in section 4.3.

- External validity – External validity is challenged when research cannot be extended to numerous real problems. We cannot ensure that the adopted standard encompasses variations of real problems from the software development domains. This is the main threat. We chose a standard dataset of 11 software projects to mitigate this threat. To the best of our knowledge, the benchmark dataset of 11 software projects is one of the most feasible and extensive datasets in requirement prioritisation areas; it contains detailed information on multiple requirements and their interdependence. Thus, the 11 software projects dataset is often used for assessments derived from initiated and developed software projects. To improve external validity, ARPT should be evaluated in additional software developments from companies involved in the varieties of software products implementation.

- Conclusion validity – Threats to the conclusion concern the relationship between the procedure and the result. The threats, in this case, originate from the comparison with optimisation algorithms. Numerous methods for requirement prioritising are compared. However, we can still not contrast ARPT to all these methods due to various factors, including the inaccessibility of these techniques' source code to the public. To minimise this threat, we compared the performance of the proposed ARPT method to that of the optimisation algorithms that are most relevant to ARPT, as these selected algorithms were tested in terms of mean squared error and error cost using the same 11 software projects dataset as the current study. Furthermore, like ARPT, these evaluated algorithms conduct their method procedures during requirement prioritisation. To state of the art, the algorithms and comparative standards indicated above are the best findings provided so far using the 11 software projects dataset.

6. Conclusion and Future Work

In the case of text features, the Adam Algorithm needs to converge as intended, especially when the feature frequency needs to be higher in the dataset. This study proposes the

ARPT algorithm to assign priority to software requirements to address this limitation. The hypothesis is tested by applying the proposed ARPT method to the software projects. The resulting priorities assigned to requirements are compared in terms of mean square error and cost error of prioritising to the requirements priorities generated by existing techniques and the ground truth. The ground truth is a complete and prioritised list of requirements and requirement dependencies for all 11 software projects. One notable advantage of the ARPT algorithm is that it outperforms Adam Algorithm on text datasets and considers the requirements' dependency. In addition, this paper systematically compares existing optimisation algorithms and the ARPT method for requirement prioritisation in terms of error cost and mean squared error. The ARPT algorithm has been evaluated on 11 software project requirement datasets, including functional and non-functional requirements such as usability, performance, etc. Compared to state-of-the-art optimisation algorithms, the experimental results showed that the ARPT algorithm outperforms the Adam algorithms and achieves a cost error of 0.0001 and a mean squared error of 1.29 at epoch 1000 for requirement prioritisation. The ARPT algorithm takes only 1.99ms to execute. The ARPT algorithm has a drawback that it does not consider requirement conflicts. In the future, we will attempt to solve this problem, and the proposed method will be evaluated on various datasets from various software domains.

The future work will also include the development of a framework to automate the classification and prioritisation of requirements, as well as considering conflicting requirements.

Author Contributions

The paper background work, conceptualization, methodology, dataset collection, implementation, result analysis and comparison, preparing and editing draft, visualization have been done by first author. The supervision, review of work and project administration, has been done by second author.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] J. Dick, K. Jackson, and E. Hull, "Requirements Engineering," 3rd edn. Springer International Publishing, pp. 1–32, 2017. https://doi.org/10.1007/978-3-319-61073-3

[2] P. Talele and R. Phalnikar, "Machine learning-based software requirements identification for a large number of features," International Journal of Computational Systems Engineering, Vol. 6, pp. 255–260, 2021. https://doi.org/10.1504/IJCSYSE.2021.123553

[3] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," Proceedings of the Conference on The Future of Software Engineering, pp. 35–46, 2000.

https://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

[4] M. Batra and D. A. Bhatnagar, "Requirements Prioritization: A Review," International Journal of Advanced Research in Science, Engineering and Technology, Vol. 3, no. 11, pp. 2350–0328, 2016. http://www.ijarset.com/upload/2016/november/6_IJARSET_monabatra.pdf

[5] R. Devadas and N. G. Cholli, "Multi aspects based requirements prioritization for large scale software using deep neural lagrange multiplier," 2022 International Conference on Smart Technologies and Systems for Next Generation Computing, pp. 1–6, 2022. https://ieeexplore.ieee.org/document/9761298

[6] A. Hudaib, R. Masadeh, and A. Alzaqebah, "WGW: A Hybrid Approach Based on Whale and Grey Wolf Optimization Algorithms for Requirements Prioritization," Advances in Systems Science and Applications, Vol. 02, pp. 63–83, 2018.

https://doi.org/10.25728/assa.2018.18.2.576

[7] R. V. Anand and M. Dinakaran, "Handling stakeholder conflict by agile requirement prioritization using Apriori technique," Computers & Electrical Engineering, Vol. 61, 2017, pp. 126–136. https://doi.org/10.1016/j.compeleceng.2017.06.022

[8] D. Singh and A. Sharma, "Software Requirement Prioritization using Machine Learning," SEKE, 2014, Vancouver, Canada, pp. 701-704, July 2014.

[9] B. B. Jawale, G. K. Patnaik, and A. T. Bhole, "Requirement Prioritization Using Adaptive Fuzzy Hierarchical Cumulative Voting," IEEE 7th International Advance Computing Conference (IACC), pp. 95–102, 2017, https://doi.org/10.1109/IACC.2017.0034.

[10] F. Shao, R. Peng, H. Lai, and B. Wang, "DRank: A semi- automated requirements prioritization method based on preferences and dependencies," Journal of Systems and Software, Vol. 126, pp. 141–156, 2017. https://doi.org/10.1016/j.jss.2016.09.04

[11] P. Talele and R. Phalnikar, "Classification and Prioritisation of Software Requirements using Machine Learning - A Systematic Review," in 11th International Conference on Cloud Computing, pp. 912–918, 2021. https://doi.org/10.1109/Confluence51648.2021.9377190

[12] P. Talele and R. Phalnikar, "Multiple correlation based decision tree model for classification of software requirements," International Journal of Computational Science and Engineering, Vol. 26, No. 3, pp. 305-315, 2023. https://doi.org/10.1504/IJCSE.2023.131502

[13] D. Greer and D. W. Bustard, "SERUM - Software Engineering Risk: Understanding and Management," The International Journal of Project & Business Risk, Vol. 1, no. 4, pp. 373–388, 1997.

[14] Q. Ma, "The effectiveness of requirements prioritization techniques for a medium to large number of requirements: a systematic literature review," Auckland University of Technology. (Doctoral dissertation, Auckland University of Technology), 2009.

[15] R. V. Anand and M. Dinakaran, "WhaleRank: an optimisation based ranking approach for software requirements prioritisation," International Journal of Environment and Waste Management, Vol. 21, pp. 1–21, 2018. https://doi.org/10.1504/IJEWM.2018.091307

[16] R. Masadeh, A. Alzaqebah, and A. Hudaib, "Grey Wolf Algorithm for Requirements Prioritization," Modern Applied Science, Vol. 12, 2018. https://doi.org/ 10.5539/mas.v12n2p54

[17] F. Hujainah, R. Bakar, and A. B. Nasser, "SRPTackle: A semi- automated requirements prioritisation technique for scalable requirements of software system projects," Information and Software Technology, Vol. 131, pp. 106 501– 106 501, 2021. https://doi.org/10.1016/j.infsof.2020.106501

[18] P. Avesani, C. Bazzanella, A. Perini and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," 13th IEEE International Conference on Requirements Engineering RE'05, pp. 297–305, 2005. https://doi.org/10.1109/RE.2005.30

[19] F. Hujainah, R. Bakar, M. A. Abdulgabber, and K. Z. Zamli, "Software Requirements Prioritisation: A Systematic Literature Review on Significance, Stakeholders, Techniques and Challenges," IEEE Access, Vol. 6, pp. 71 497–71 523, 2018. https://doi.org/ 10.1109/ACCESS.2018.2881755

[20] P. Talele and R. Phalnikar, "Software Requirements Classification and Prioritisation Using Machine Learning," Machine Learning for Predictive Analysis, pp. 257– 267, 2021. https://doi.org/ 10.1007/978-981-15-7106-0_26

[21] S. Ruder, 2016. [Online].

[22] Available: https://arxiv.org/pdf/1609.04747.pdf

[23] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," International Conference on Learning Representations, 2015. https://doi.org/10.48550/arXiv.1412.6980

[24] S. Vijayakumar and S, "Use of Natural Language Processing in Software Requirements Prioritization - A Systematic Literature Review," International Journal of Applied Engineering and Management Letters, pp. 152–174, 2021. https://doi.org/10.47992/IJAEML.2581.7000.0110

[25] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," Journal of Machine Learning Research, Vol. 12, pp. 2121–2159, 2011. https://jmlr.org/papers/v12/duchi11a.html

[26] [Online]. Available: https://www.coursera.org/lecture/deep-neural-network/rmsprop-BhJlm

[27] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén, "Experimentation in Software Engineering," Springer Publishing Company, Incorporated, 2012. https://doi.org/10.1007/978-1-4615-4625-2

[28] Prof. Barry Wiling. (2018). Identification of Mouth Cancer laceration Using Machine Learning Approach. International Journal of New Practices in Management and Engineering, 7(03), 01 - 07. https://doi.org/10.17762/ijnpme.v7i03.66

[29] Karthick, S. ., Shankar, P. V. ., Jayakumar, T. ., Suba, G. M. ., Quadir, M. ., & Paul Roy, A. T. . (2023). A Novel Approach for Integrated Shortest Path Finding Algorithm (ISPSA) Using Mesh Topologies and Networks-on-Chip (NOC). International Journal on Recent and Innovation Trends in Computing and Communication, 11(2s), 87–95. https://doi.org/10.17762/ijritcc.v11i2s.6032

[30] Sharma, R., & Dhabliya, D. (2019). A review of automatic irrigation system through IoT. International Journal of Control and Automation, 12(6 Special Issue), 24-29. Retrieved from www.scopus.com