

A Machine Learning Algorithm Based on Inverse Problems for Software Requirements Selection

ABSTRACT

Software requirements selection is one of the main software engineering processes which have to be taken into account by software companies to determine what will be included in the next release of a product in such a way that the highest possible number of stakeholders get satisfied while the cost is kept at a minimum for the company. Minimizing the total cost of adding a new feature in the next release of a package and maximizing overall customer satisfaction are contradictory objectives. In this paper, first, an adaptive multi-objective prioritization model is discussed. Then we describe how discrete inverse problems ideas can in fact be formulated to obtain an “Added Degree of Importance” (ADI) function of client requirements which could be used to classify and prioritize the software requirements for next release. The numerical implementation of the proposed model with a case study on software requirements selection shows the effectiveness of the multi-objective inverse model (IM) approach. The results of the implemented model have been compared with some of the recent relevant models. Main future of the model is that it has been designed by the assignment of a real score for each of the requirements unlike just classification provided in the literature.

Keywords: Software requirements prioritization; multi-objective evolutionary algorithm; next release problem; inverse problems; machine learning

1. INTRODUCTION

Software has become ubiquitous in present society and thus it has become important to avoid and detect any development disruptions. New software product development is commonly risky, judging by the number of high-profile failures that continue to occur-especially in software engineering. We can trace many of these failures back to requirements-related issues. Software requirements selection is a very challenging topic in software engineering and has been the subject of active research for many years. We can observe exponential growth of complexity in software systems in the last decade. Moreover software

products have to be developed on time and on schedule. Thus, one of the first issues which have to be taken care of by the software companies is to determine the requirements for the next release and prioritize requirements, in such a way that the highest number of customers get satisfied with minimum cost for the company. In most of the cases, it is not possible to develop all the features requested by the client. So prioritization of requirements is an important task in software engineering with agile methodologies. In agile development, the software product is developed in incremental releases which have to be produced and delivered to customer in a short time frame. A new set of requirements is proposed for each iteration. Therefore it is very important to establish the priority of the requirements based on many criteria. For this purpose, we use a multi objective inverse approach algorithms as it can manage conflicting objectives and eliminates the need of artificial adjustments included in traditional single objective optimization models.

Due to increased complexity and constraints such as limited resources, limited time, and limited technical capabilities on software products, minimizing production effort when implementing large amount of requirements becomes very crucial step in software development. In some cases all the new features suggested by the clients may not be possible to develop ([1] and [2]). Therefore software requirements prioritization is especially important with the incremental approaches of software development, such as in the agile methodologies [3], [4], and [5].

Software requirements represent a way of interconnecting any pair of requirements by means of some links of interactions. Thus, it is quite natural that interaction structure of software requirements can be represented in a simplified form. When we have a problem of choosing next requirement to implement from a set of requirements, it is desirable to work with a number of known requirements that has been already implemented as core part of software since the software requirements may be very large and its interaction structure too complicated [6].

2. PROBLEM DEFINITION

The aim of this project is to present an efficient approach for software requirements evaluation and scoring. It is a crucial part for software project development because the highest possible number of stakeholders must be satisfied while keeping the cost at a minimum for the company. The high margin of accuracy in the system is very necessary and quite beneficial for stakeholders. This paper analyses the software requirements data and provides a framework which prioritize the requirements in two phases as shown in Figure [1]. This project focuses on the use of knowledge discovery methods thru inverse machine learning algorithms with soft computing methods to suggest a solution.

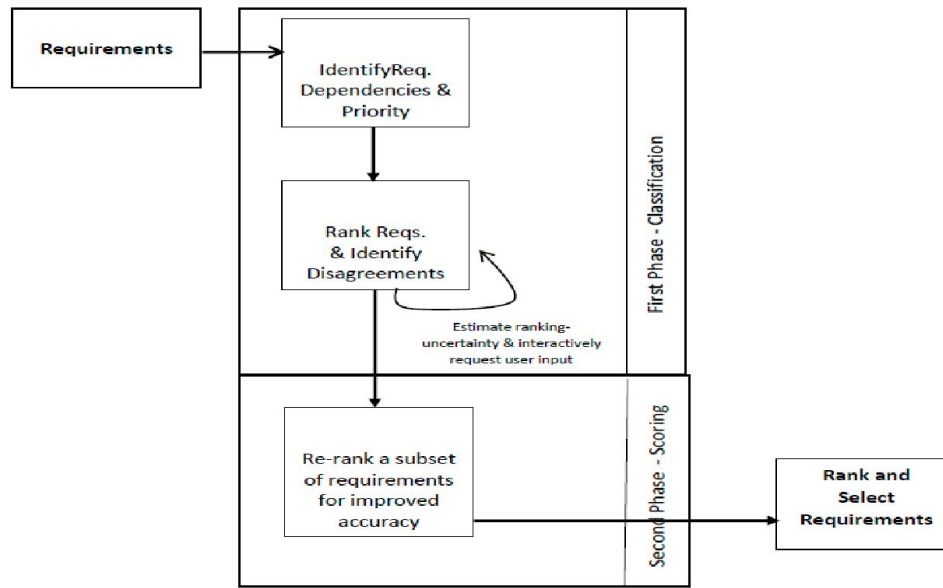


Fig. 1. Two phases of IML Model

Objective of the process must be to maximize the number of stakeholders who are satisfied and minimize the cost for the company. To minimize loss from the company's point of view, the company needs a choice lead with respect to what to give endorsement of the advance for implementation and what not to. A software requirement's attributes are considered by developers before a choice is taken in regards to its implementations. The literature review indicates that the following attributes need to be considered: requirements' dependencies, requirements' quality, stakeholder's preferences, development risk, development cost, business value, resources, and time restrictions.

Evaluating a requirement is an important challenge in software engineering. In some mid-sized projects, we can get quite a large number of requirements [7]. It is a very complicated task to decide whether signing off on a requirement for implementation would be beneficial or not [8]. The aim of every method is to classify the bad requirements with the good ones. It becomes very hard for the manager to decide just by inspecting the application form with some details [9], [10]. Therefore, software requirements prioritization is one of the main phases in software development.

The advancement in the information technology has led companies and developers to electronically store all the details of the requirement. This led to the urgency of making a system which can automate software requirements prioritizing decision so that the stress on managers can be reduced and efficiency could be improved. There are lots of methods which have been proposed and covered in the literature ([11], and [12]) using supervised and unsupervised learning. Most of these methods have used different classifying techniques with reasonable accuracy but no one has explained the details of the classification procedures. As it plays a crucial role in the prioritizing procedures so there must be a proper justification of the approval and rejection of a specific requirement.

Similar problems have been previously studied by a number of researchers in the domain of the reconstruction of classification ([13], [14], [15], [16], [17], [18], [19], and [20]). But these studies did not go beyond classification. In this study, we associate each requirement with a real score which gives richer interpretation. It enables managers to take effective decision and prioritize the requirements.

3. MAIN ANALYSIS

As we implemented the algorithm in a multi-objective decision support system framework, the parameter recovery of our proposed Inverse Machine Learning (IML) model, described below, is established with respect to the quality of the ADI parameter in each experiment. In this study, we did use the Aggregated Degree of Importance (ADI) indicators in such a way that we may conduct a comparative study with other recently published papers. The model measures the aggregated degree of importance indicator of each requirement with respect to the two objectives which are low development cost and high overall clients' satisfaction. Note that requirement interactions are taken into consideration in the IML model as well.

IML is based on a non-linear learning function. In most models, the time to train the data is relatively long and also susceptible to local minimum traps. But this can be removed or mitigated by doing multiple rounds and picking the best learned model. We use "createDataPartition" function in "caret" R-package to split original data into training data and testing data by informing the ratio of training data. The package make training process for complex regression and classification problems easier for users. We built a model for software requirements prioritization as described in Algorithm 1 by using our IML algorithm, while 80% of the data are training data, and the other 20% of the data as testing data.

Algorithm 1:

```

1: RIR generateRandomInitialRequirements(R –training set)
3: ADI  $\varnothing$  (Initialize the ADI function)
4: while not stop condition satisfied do
5:   for  $i = 1$  to R_Size do
6:      $x_{target} \leftarrow R_i$ 
7:     /* Select 80 random requirements from R.
 $x_{r1} \neq x_{r2} \neq \dots \neq x_{r80}$  */
8:      $x_{r1}, x_{r2}, \dots, x_{r80} \leftarrow$  select random requirements (R)
10:    for each requirements in  $x_{training}$  do ( $\sigma = \frac{1}{n}g(\frac{K}{n})F$ )
12:       $\sigma_{training}[r] \leftarrow g[x_{r1}, x_{r2}, x_{r80}]$ 
16:    end for
17:     $\sigma_{trial}$  evaluateIndividual ( $\sigma_{trial}$ )
18:     $R_i$  IM( $\sigma_{target}, \sigma_{trial}$ )
19:  end for
20:
21:
22: end while
23: ADI function  $\sigma_{target}$ 
24:   for  $x_{target} \neq x_{trial}$ 
25:     if  $ADI[x_{trial}] < ADI[x_{trial+1}]$ 
26:       temp  $ADI[x_{trial}]$ 
28:        $ADI[x_{trial}]$   $ADI[x_{trial+1}]$ 

```

```

29:         ADI[ $x_{trial+1}$ ] temp
30:     end if
31: end for
33: return  $x_{target}$ 

```

IML is a supervised learning problem, where we use the training data sets to predict a target function. The basic and detailed frame work for the IML model is provided in Figure 2 and Figure 3 respectively. In IML supervised learning we predict an outcome based on a set of inputs provided by the user or mentioned in the data file. For example, a common model is a linear model, where the prediction is given by a linear combination of weighted input features. The output from the prediction might have different importance and value based on the task at hand. For example, it can also be used as a ranking score when we want to rank the outputs. The target function σ (ADI) are the undetermined part that we need to learn from data. We need to find a way to find the best target function, ADI, with best parameters given the training data. In order to do so, we need to define a so-called ADI indicator function, to measure the performance of the model given a certain set of parameters for the target function. ADI indicator function σ is computed by solving an inverse problem with similar software requirements on the next release of agile development. Therefore the value of ADI indicator σ returns known software requirements when substituted into Eq. (1). That is, each software requirement has a corresponding ADI indicator which can be used to find approximate a new set of Software requirements for next release.

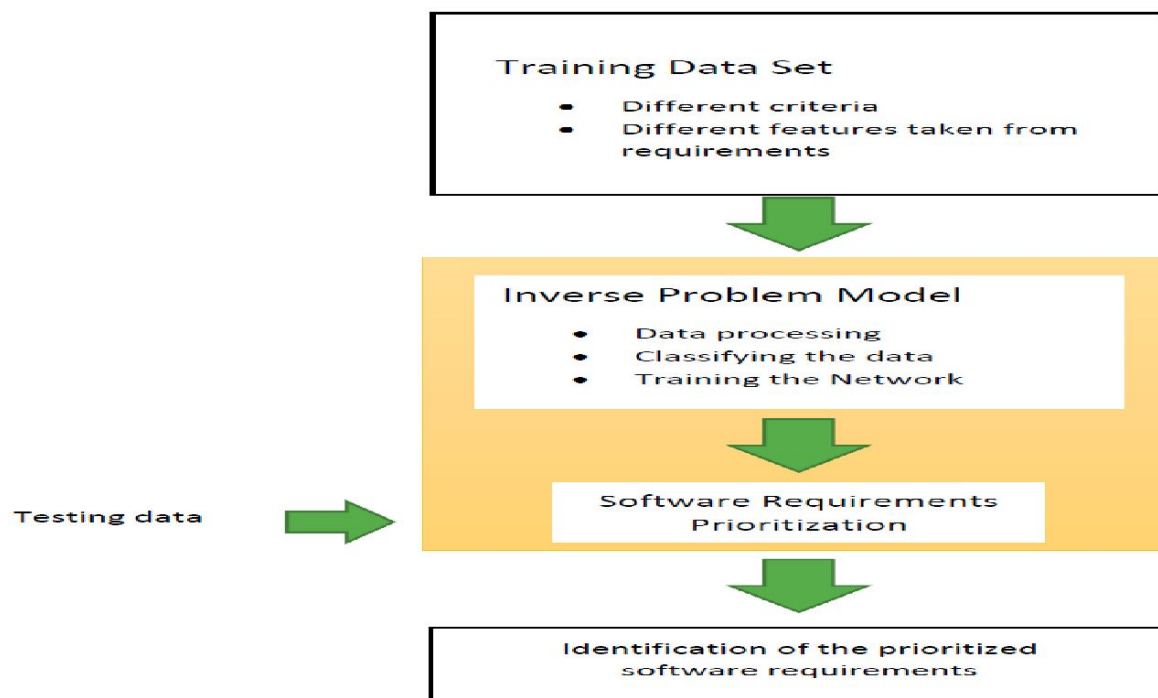


Fig 2. Basic Architectural framework of IML Model

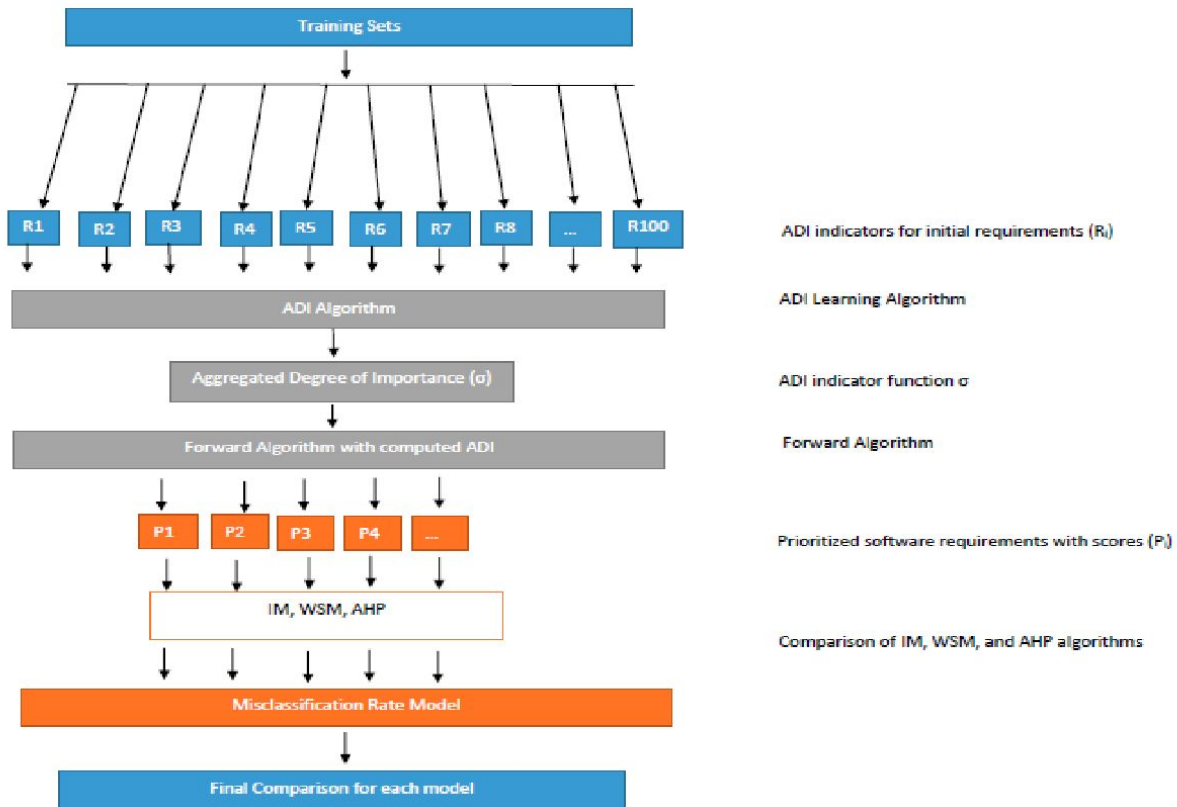


Fig. 3. Detail Architectural framework of IML Model

In the next release problem, one has to answer the question of “how to identify which requirements should be developed and delivered taking into consideration multiple complex factors and assuming that a relationship between a clients’ requirements and some of its attributes exists”. To achieve this sufficiently early in software development, it is essential to count on a solid model that can allow early detection of disruptions to the software development. The problem of producing best possible requirements can be translated into a suitable supervised inverse problem. The applied tool that we consider is based on the use of discrete version of adaptive machine learning approach in which a network of software requirements consists of interconnecting any pair of requirements by means of some links. The problem of discovering the prioritized software requirements of all the possible software requirements from a collection of implemented requirements for the next release problem can be seen as a type of inverse problem, analogous to those arising in tomography, but in a discrete form. Chung and Bernstein’s work [21] led to a key result in the domain of discrete inverse problem which proves the uniqueness of weights (or connectivity) under certain monotonicity conditions. This problem may be considered as an inverse problem of finding the $\sigma = \sigma(x) > 0$ in

$$Af = F \quad (1)$$

where X and Y be two Hilbert spaces in $\mathfrak{H}^2$, $Af = f$, $A : X \rightarrow Y$ a linear bounded operator, and $F \in X$, σ is a reconstruction of ADI of the software requirements function. The solution $u(x, y)$ is given for $x \in X$, $y \in Y$, and $cl(X) \cap cl(Y) = \emptyset$. The formulation of Eq. (1) is given in [22]. In Eq. (1), u is the software requirements for the next release problem. That is what we are trying to determine in X based on known implemented requirements in in *subset of* Y . A value of σ is required before Eq. (1) can be used. ADI indicator is computed by solving an inverse problem with similar software requirements. Therefore ADI indicator is that the value of σ , when substituted in to Eq. (1), yields known software requirements.

Therefore each software requirement has an associated ADI indicator, and one such value can be used to determine the best possible approximation of a software requirements on the same underlying.

The linear inverse problem associated to (1) can be stated as “Recovering the function f satisfying the eq. (1), given A and F “. In general, this problem is ill-posed. One may achieve existence and uniqueness by introducing the generalized solution f defined as

$$\min_{f \in X} \|Af - F\|_Y^2 \quad (2)$$

Answer to the question of existence of the generalized solution f depends on the condition $TF \in \text{Range}(A)$, where T is the projection on the range of operator $\hat{A}$. Since the generalized solution f does not depend continuously on the F , so is not known, but only a *noisy* $F_\sigma \in Y$ is given, where $\|F - F_\sigma\|_Y \leq \sigma$, one may use well known regularization model to find a solution depending continuously on the data by replacing (2) with the following

$$\min_{f \in X} \{ \|Af - F_\sigma\|_Y^2 + \|f\|_X^2 \}, \quad (3)$$

and

$$f_\sigma = (A^*A + I)^{-1}A^*F_\sigma \quad (4)$$

where $\alpha > 0$.

An important issue is the crucial choice of the regularizations parameter α as a function of the noise. The reconstruction error

$$\|f_\sigma - f\|_X \quad (5)$$

must be made small by selecting proper α as a function of the noise level σ . By assuming that the generalized solution exists we may end up with the following measuring tool for learning process

$$\|Af_\sigma - TF\|_Y = \|Af_\sigma - f\|_Y \quad (6)$$

which will be used in our analysis of learning. Here notice that the generalized solution may not exist.

Notice that we can easily extend this to the case of a noisy operator $A_\sigma : X \rightarrow Y$ where

$$\|A - A_\sigma\| \leq \sigma,$$

and $\|\cdot\|$ is the operator norm. To deal with the problem of learning in the framework of ill-posed inverse problems we have to define a direct problem. Also the nature of the noise σ in the context of machine learning is not clear. First, we need to show how to deal with the problem of learning in the ill-posed inverse problem setting. To do this, we have to define a direct problem by means of suitable operator $A : X \rightarrow M \subset X$ such that

$$(Af)(x) = \langle f, X_M \rangle_Y,$$

where K is the Riesz type kernel of X . Since K is bounded, A is a bounded linear operator as well [IS]. That is simply

$$(Af)(x) = f(x) \quad \forall x \in M, f \in X,$$

It is easy to check that

$$\frac{1}{n} \sum_{i=1}^n (f(x_i) - yF_i)^2 = \|A_X f - yF\|_X^2$$

Therefore the estimator function f_σ is the Tikhonov regularized solution of

$$A_X f = F. \quad (7)$$

This is easy to show by considering the operator $A_X : H \rightarrow R^n$ defined by $(A_X f)_i = f(x_i)$, $i = 1, \dots, n$ and

$$L(F_i, f(x_i)) = \sum_{i=1}^n (yF_i - f(x_i))^2 \quad (8)$$

In such a way that

$$\|A_X - y\|_n^2 = \frac{1}{n} \sum_{i=1}^n (F_i - f(x_i))^2 \quad (9)$$

where the norm is $1/n$ times the Euclidean norm on R^n and $y = (y_1, \dots, y_n)^T$. This shows that the solution f_σ of the minimization problem (4) with the function (8) is equal to the Tikhonov regularized solution of the linear discrete inverse problem $A_X f = y$.

Now one can define the explicit form of the solution f_σ as follows:

$$f_\sigma(x) = \sum_{i=1}^n \alpha_i K(x, x_i), \quad \sigma = (K + nI)^{-1}y, \quad (10)$$

where I is identity, K is the kernel defined by $(K)_{ij} = K(x_i, x_j)$, $y = A^*F$ and $\alpha = (\alpha_1, \dots, \alpha_n)^T$.

The parameter σ is used to better control stabilization of the inversion operation in (10). This problem is not stable. By following the generalizations approach provided in [22], we may define

$$f_\sigma(x) = \sum_{i=1}^n \alpha_i K(x, x_i), \quad \sigma = \frac{1}{n} g\left(\frac{K}{n}\right)F \quad (11)$$

where $g : M_n(R) \rightarrow M_n(R)$. Notice that $M_n(R)$ represents the set of $n \times n$ matrices with real entries. Since this inverse problems is "ill-posed", the choice of g can very much affect the sensitivity of computations. A better approach is to think of ADI function σ as a function rather than a constant. An inverse problem of best requirements prediction is to determine the distribution of expected client

requirements implied by implemented client requirements in the previous releases. In order to compute the aggregated degree of importance indicator of a requirement, a weighted sum of individual requirement's goodness indicators is computed. Importance of each indicator can be determined according to the results of empirical studies on practical importance of different requirement selection criteria. The model presented below uses multiple requirement selection criteria although other criteria mentioned in literature can be incorporated, if necessary. The main difference of the proposed model from the traditional requirements selection models is that ADI indicator is not specified by discrete data points and these data points are not assigned to the specific location. Instead of that the ADI indicator approximates the ADI indicator parameter around the implemented requirements, and requirements with ADI indicator are picked for the next release. In this study, we propose a multi-objective decision support system presented in Figure 4 and 5 for software requirements selection for next release problem based on a machine learning regularization of an inverse problem.

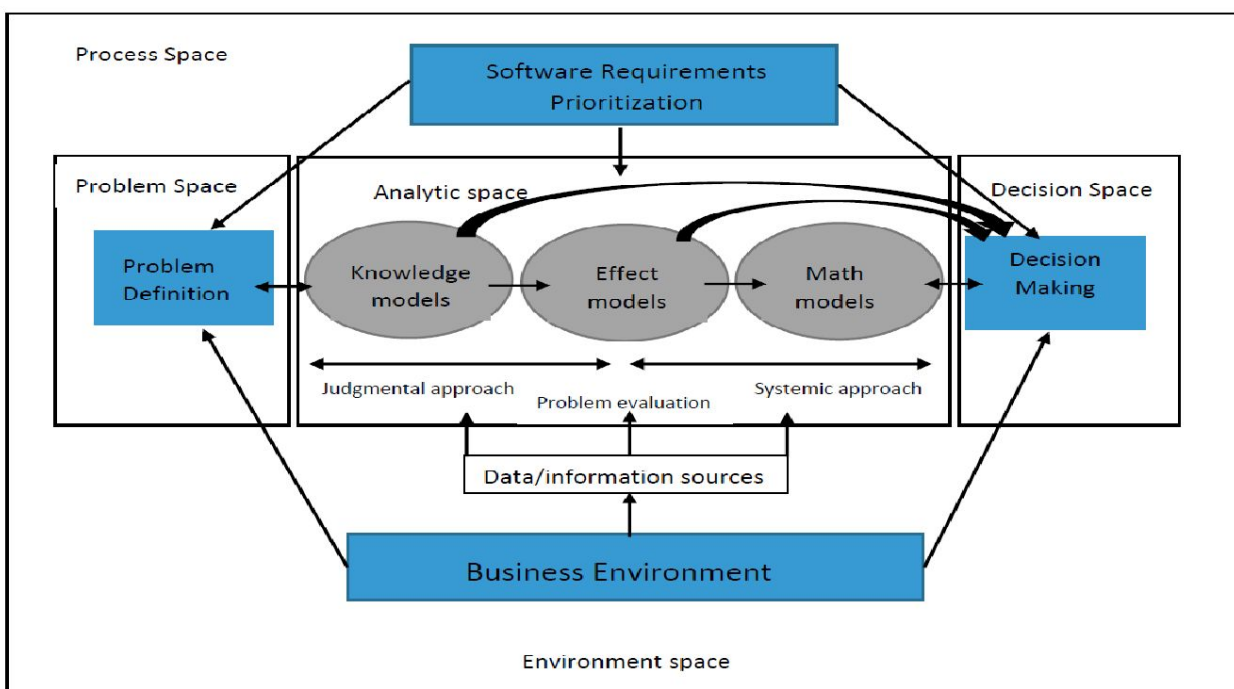


Fig. 4. A general approach of requirements evaluation using mathematical modeling

Figure 4 show the general approach that is followed before assessing the IML method. The training dataset which consist of different features and attributes about a requirement like technical precedence, conflicting priorities, stakeholder priorities are fed to the network, before the data is processed (line 1-line 8 in Algorithm 1).

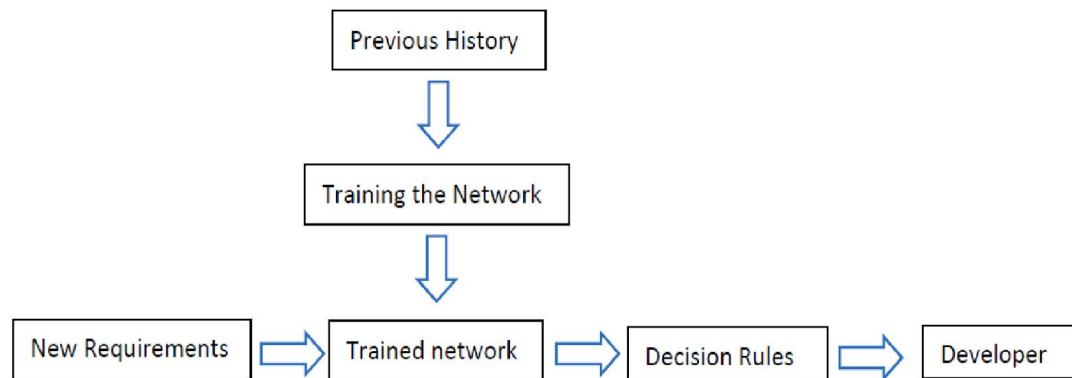


Fig. 5. Core architecture of the system

For computational reasons, we suggest to approximate this nonlinear problem by a linear one so that we may split the inverse problem into two parts. First part is a simpler, ill-posed problem (an integral equation with a Riesz-type kernel) problem and the second part is a well-posed problem. This way we can control the propagation of errors due to the ill-posedness.

$$(aI + \Theta) \sigma(y) = A^*F \quad (12)$$

where $A^*F(y) = \int_{\Omega^k} F(x)/|x-y|^2$, $y \in Y$ for some regularization parameter $\alpha > 0$. Notice that the

linearization method led us to a Fredholm integral equation of first kind with Riesz kernel. Since this equation represents an ill-posed problem, we needed some kind of regularization method to overcome this difficulty. We used the Tikhonov regularization method and, by discretizing the regularized equation (12), ended up with the system of linear equations.

4. PROPOSED SOFTWARE SYSTEM

The aim of every method is to classify the bad requirements with the good one. It becomes very hard for the software manager to decide just by inspecting the requirement form with some details. A variety of soft computing methods are being adopted for it. Svahnbergan and Goreschek in 2010[23] used GA to program the models. In 2014 Sagrado & Aguila[24] and Svensson & Groschek[25] in 2011 used NN for prioritize the requirements, which gives reasonable accuracy rates. Kumari and Shrinivas [26] in 2013 used adaptive neuro-fuzzy models for the analysis of software requirements system and shows its significance as compared with other techniques. There is indeed a need of an automatic system which can assist the developers to help them in requirement approval.

As the modern business deals with lots of complexity like rapidly changing situations, sudden crises, managers are trying to assist additional tools to help them in predicting and managing the situation effectively. Hence soft-computing methods like neural network, fuzzy, genetic and their hybridizations like, adaptive neuro fuzzy are being used efficiently. These methods are being used for clustering, modeling, forecasting, and classifying the available data. The aim of the study is to develop a requirements prioritizing system using inverse machine learning algorithms developed here so that all requirements are prioritized and implemented based on multiple criteria as part of a system. The component diagram of the system is shown in Fig. (6).

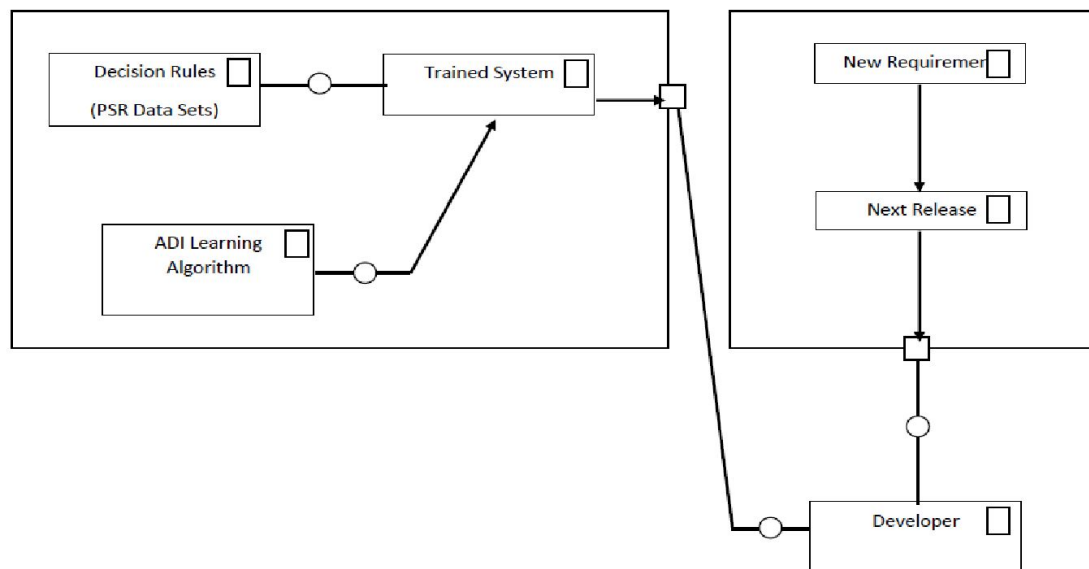


Fig. 6. Component diagram

The whole system is depicted in Fig. 7. The first phase of the whole system is the business understanding. That emphasizes on understanding the project objectives and requirements from a business point of view, and converting this knowledge into a software requirement prioritization problem definition, and an initial plan designed to achieve the objectives.

The data preparation phase may include some specific activities to make sure data is ready for implementation phase and this may be performed in an iterative fashion. Attribute selection needs to be performed at this level.

In the phase of modeling, the inverse problem modeling techniques are selected and applied, and their parameters are adjusted to optimal values. As seen in Fig. 7, this phase is also iterative in nature.

At the evaluation phase, we built a required model that satisfies the key objective of software requirements prioritization and decision on the software requirements prioritization results should be reached.

The last phase is the deployment. Since the purpose of the model is to prioritize the requirements, the knowledge gained on requirements will need to be organized and presented in a way that is useful to the stakeholders. It is important for the stakeholder to understand the steps that need to be carried out in order to actually make use of the models developed.

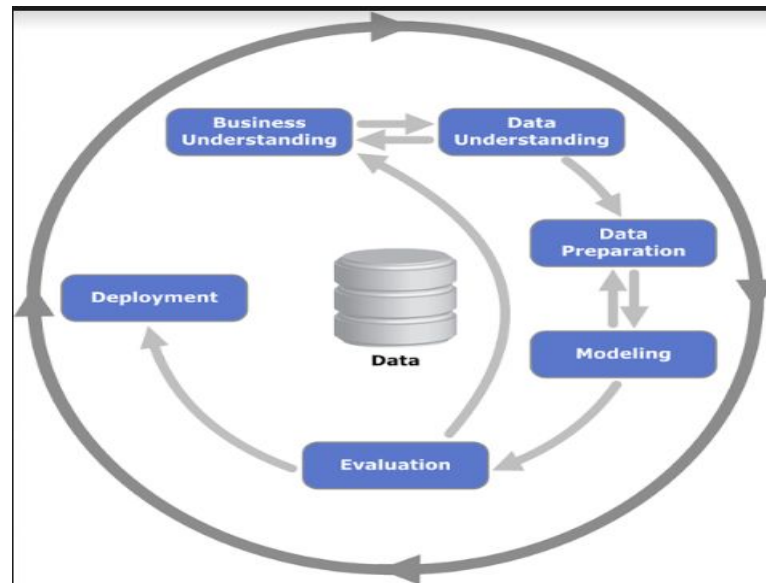


Fig. 7. Business model diagram

Knowledge discovery is finding the patterns from a large set of requirements and extracting useful knowledge and information from it. It involves discovering patterns from the large set of requirements using different mathematical models. It is formally the intersection of Machine learning (ML) and artificial intelligence with statistics and data base system. There are different mathematical models available like Neural networks, fuzzy, genetic algorithm, support vectors along with their hybridizations extract knowledge from the data bases ([27], [28], [29]). These soft computing methodologies are widely accepted for classification, clustering and predictions.

5. RESULTS AND DISCUSSION

First, we start by describing the approach used in the experiments and the data sets used to measure the performance of the proposed algorithm. Next, we discuss the obtained results by using different quality measures and compare them with some of the results obtained by other studies. To show practicability of the approach, we study a sample software project with 100 requirements. In Classification phase 100 data points were used for training and 20 for testing. Each requirement is associated with a data point that consists of four real-valued attributes; cost, stakeholder preference, effort, and dependencies.

Each requirement has an associated effort estimate score between 1 and 10. Effort sizes may be different for different increments. Five stakeholders were used to score the 100 requirements with priority scores from 1 to 5. These scores are shown in Fig 7. As we can see, different stakeholders in some cases assign more or less the same priority to requirements. . Each requirement has a list of all the clients that rated the requirement with the associated rating score. In a similar way, each stakeholder assigns a value to each requirement based on the contribution to the business value of the product by that requirement.

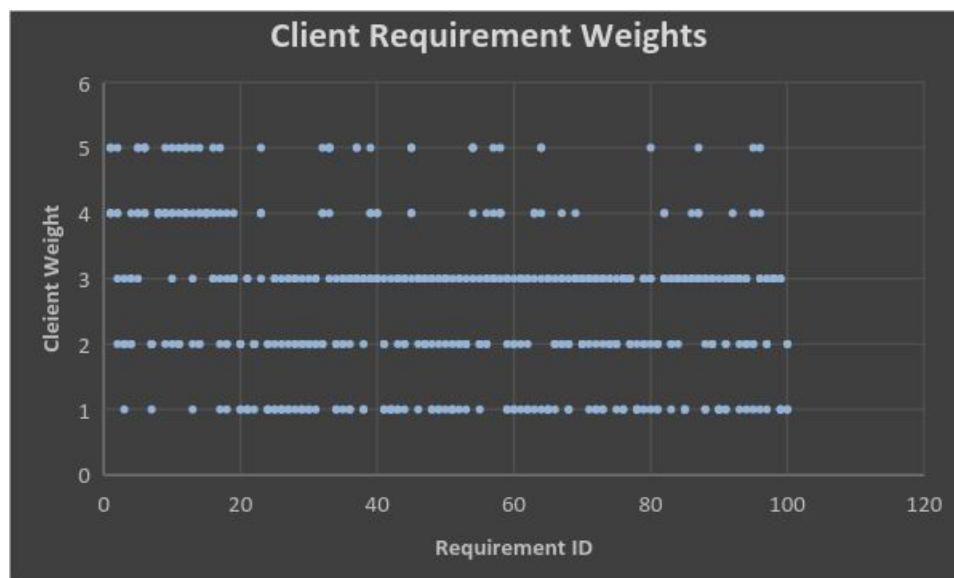


Fig. 8. Relationship between average client weights and requirements

The main model of IML is tree ensembles which differ from decision trees, where the leaf only contains decision values. In IML, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification. Regularization is used internally by IML for parallel execution of the training model.

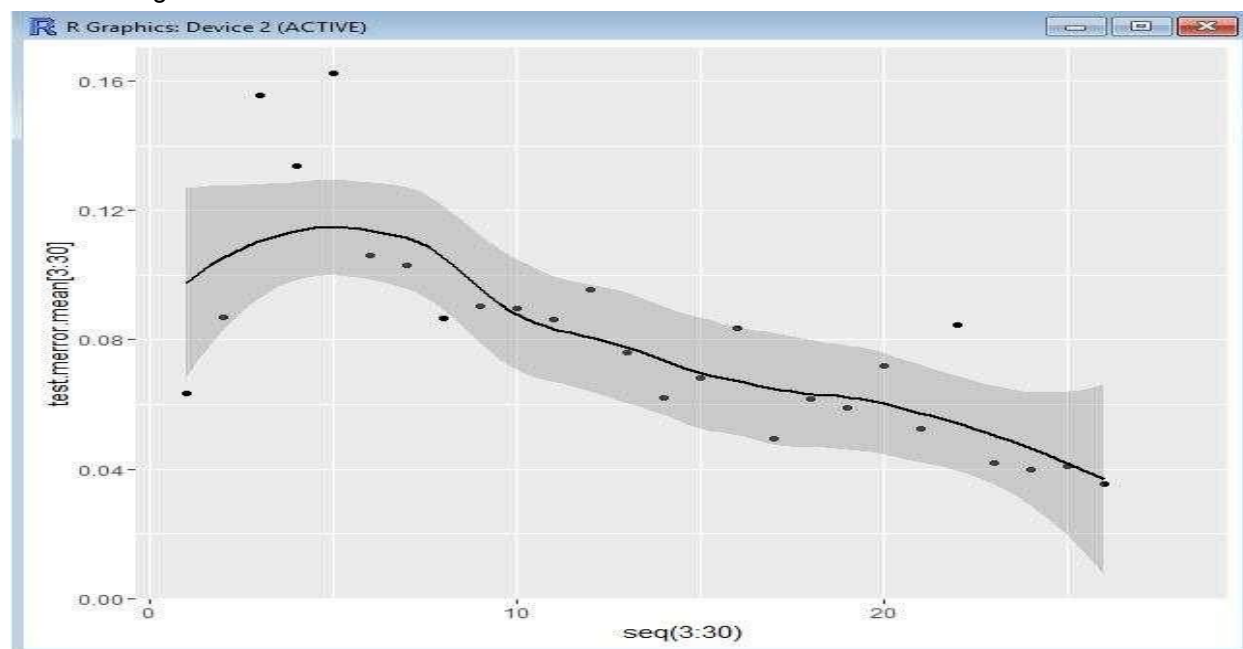


Fig. 9. Iterations vs error rate

The IML way of solving the problem will split the input dataset into training data and test data. The IML package will build and use the train data and it also uses the test data to find the accuracy with which it

can predict the priority status for those inputs. This incremental learning approach makes the adaption to new evidence possible instead of starting from scratch each time. The approach involves a key decision: when to change or adapt the model by incorporating fresher data. Our approach to come up with a best decision involves estimating accuracy by checking if the performance of our current model is starting to degrade. If we observe a sudden decrease in this performance, then we can take this as a good indication of some form of regime shift. The matching error rate is the ratio of wrong match to the total number of matches. As the number of iterations increases the error rate is going to converge to a constant. This is what has been represented in the above graph where the error tends to converge to a constant values after 30 iterations (Fig. 9).

The implementation of the IML method was evaluated using 100 requirements. The IML method was designed with special focus on scalability and the ability to prioritize a large set of requirements. To achieve this objective, the IML method prioritizes requirements in two phases. During the first phase, requirements are quickly prioritized albeit with low accuracy, essentially focusing on dividing the requirements with respect to the cutoff threshold. The first phase accounts (Fig. 10) for uncertainty and does not carefully prioritize the requirements that are close to the cutoff threshold.

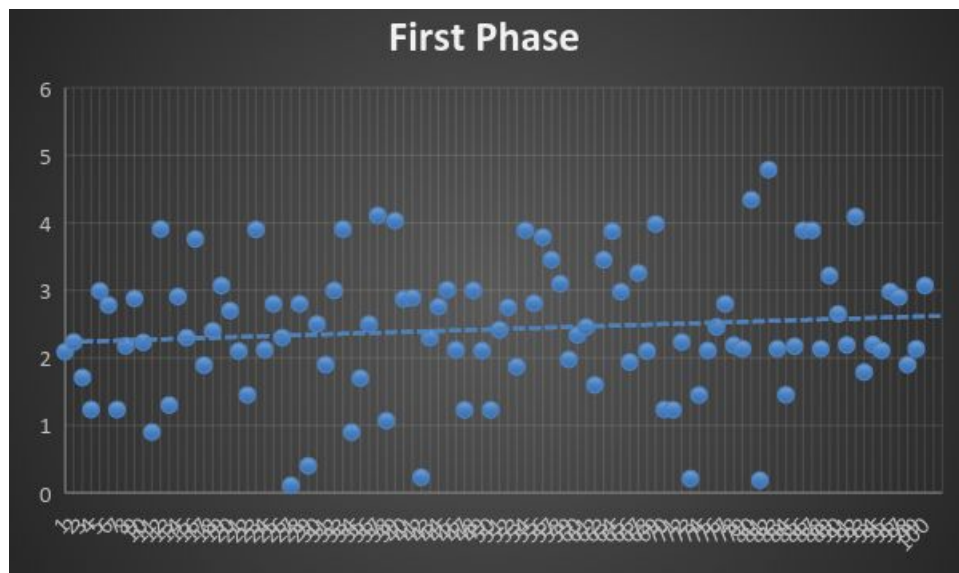


Fig. 10. First phase – Classification

The requirements with assigned real scores are prioritized for increased accuracy in the second phase by using the ADI indicator parameter around the implemented requirements (Fig. 11).

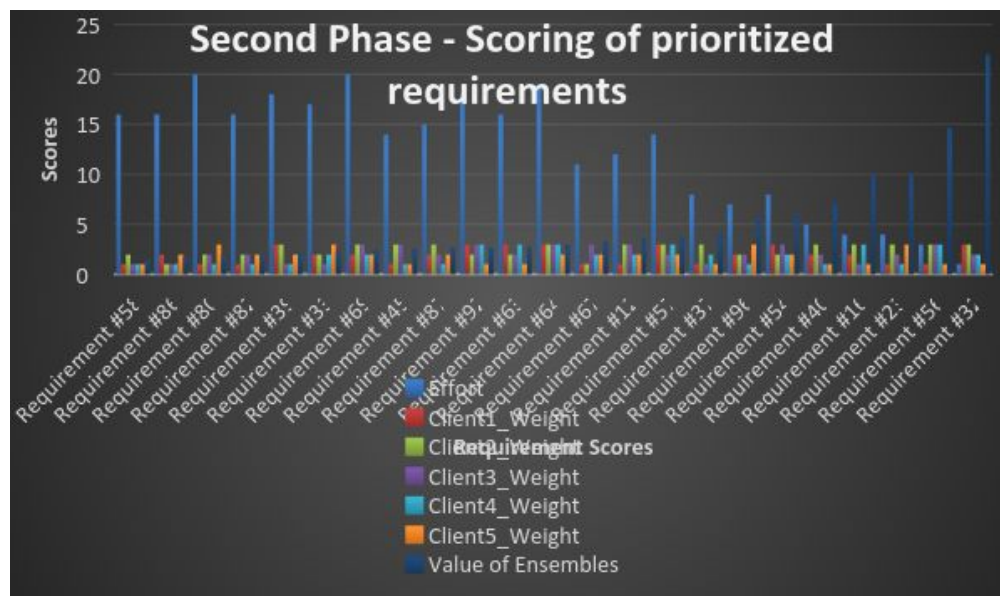


Fig. 11. Second phase – Prioritized requirements for next release problem with scoring

To verify effectiveness of suggested IML model for requirements prioritization in mid-sized systems, we have implemented and compared the results of running the IML model with MWS and AHP models. We compare our model IML with the AHP and WHS multi-criteria decision-making methods for requirements prioritization in the Table 1. Misclassification rate for each set of requirements for each model indicates that proposed IML algorithm performs much better. We also compared the accuracy of the models with or without classification in Table 2. The accuracy of IML is higher than the accuracy of AHP and WHS with classification. On the other hand, without classification, accuracy of IML is almost same as AHP, but lesser than WHS.

Description		R ²	MSE	RMSE
IML	Training	0.9717	0.009	0.0951
	Testing	0.7105	0.0648	0.2543
AHP	Training	0.8991	0.0312	0.1766
	Testing	0.7953	0.0315	0.1775
WHS	Training	0.9082	0.0271	0.1646
	Testing	0.6596	0.0779	0.2792

Table 1. Performance analysis of the improved IM, AHP, and WHS models for software prioritization

Table 2. Accuracy

Decision class	Method	Accuracy	Coverage	Real Accuracy
YES	IML	0.978	0.946	0.925

	AHP	0.952	0.932	0.957
	WHS	0.962	0.992	0.954
NO	IML	0.633	0.740	0.468
	AHS	0.644	0.850	0.823
	WHS	0.862	0.890	0.767
All Classes (YES + NO)	IML	0.964	0.935	0.901
	AHS	0.961	0.942	0.932
	WHS	0.958	0.987	0.945

Table 3. Learning time

Method	Learning time	Size of requirements
IML	801 seconds	100
AHP	616 seconds	100
WHS	647 seconds	100

Table 3 shows the learning time of each model. One can see that the learning time for AHP and WHS is shorter than for IM. The results demonstrates that the IML method is reliable and it could be used for requirements prioritization problem.

Even though the obtained results are promising and indicate that it performs well, the implementation could be improved in a number of ways. Some of those improvements may be an increasing number of software requirements and the software requirements attributes used in IML, adding an intermediary step between phase 1 and 2 to the IML to improve classification of requirements, and a better way of gathering input from stakeholders for prioritization. Therefore, future work should pay closer attention to making system modular and flexible. The implementation of IML should be converted to an expert system by fully optimizing the whole process.

5. CONCLUDING REMARKS AND FUTURE WORK

In this study we presented a clear connection between regularization theory for inverse problems and machine learning to introduce a new machine learning regularization based on an inverse problem approach. Unlike traditional classification approach, in the proposed approach, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification. Then we apply the proposed model to a software requirements prioritization problem.

The new multi-phase approach showed to be promising for the complex software requirements prioritization problem. Experimental results are showing advantages of this new approach in comparison to the standard proposed approaches.

Although we have some promising preliminary results with the proposed approach, this is definitely the beginning of more general mathematical/theoretical studies. Several other issues need to be further addressed, in particular we intend to test our algorithm with different types of software requirements. We think this study implies a new method of software design. Therefore, our long-term goal is to design and build an intelligent, accurate, flexible software prioritization system with distributed and real time characteristics. This analytical engine can be used to identify the most important requirements in

prioritized format.

Previously proposed software requirements prioritization models (e.g., collaborative cloud computing and grid computing) lead to low efficiency and high overhead or are ineffective in locating satisfying resources in an environment with a multiple number of resource attributes. In addition, most models exhibit limited flexibility by relying on a predefined attribute list for resource description and offering only limited help. This project presents an efficient and high-fidelity IML based resource Information Service (IIS). IIS constructs IML functions and relies on them to cluster the data of resources with similar attributes for efficient requirements prioritization. More importantly, it is effective in locating satisfying resources in an environment with an enormous number of resource attributes.

REFERENCES

- [1] Holbrook EA, Hayes JH, Dekhtyer A, Li W (2013) A study of methods for textual satisfaction assessment. *Empir Softw Eng* 18:139-176.
- [2] Bjarnason E, Wnuk K, Regnell B (2012) Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering. *J Inf Softw Technol* 54:1107-1124.
- [3] Jantunen S, Lehtola L, Gause DC, Dum Dum UR, Barnes RJ (2011) The challenge of release planning. In *Software product management (IWSPM), 2011 Fifth International workshop*. IEEE, pp 36-45.
- [4] Li C, Akker M, Brinkkemper S, Diepen G (2010) AN integrated approach for requirement selection and scheduling in software release planning. *Requir Eng* 15:375-396.
- [5] J. del Sagrado, I.M. del Aguila, F.J. Orellana, Requirements Interactions in the Next Release Problem, in: *Proc. Of Genetic and Evolutionary Computation Conference (GECCO 2011)*, 2011, pp. 187-188.
- [6] Aasem, M., Ramzan, M., & Jaffar, A (2015) Analysis and optimization of software requirements prioritization techniques. *International conference on information and emerging technologies (ICETET)*, IEEE Computer Society, 1-6.
- [7] Bjarnason E, Wnuk K, Regnell B (2012) Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering. *J Inf. Softw. Technology* 54:1107–1124.
- [8] Wnuk K, Regnell B, Berenbach B (2011) Scaling up Requirements Engineering – Exploring the

Challenges of Increasing Size and Complexity in Market-Driven Software Development. Requirement Eng. Found Softw. Qual. Lect. Notes Comput Sci 6606:54–59.

[9] Barney S, Aurum A, Wohlin C (2008) A product management challenge: creating software product value through requirements selection. J Syst Archit 54:576–593.

[10] Svahnberg M, Goreschek T, Feldt R, Torkar R, Saleem SB, Shafique MU (2010) A systematic review on strategic release planning models. Inf Softw Technol 52:237–248

[11] Babar, I. B., Ramzan, M., & Ghayyur, S. A. K (2011) Challenges and future trends in software requirements prioritization. International conference on computer networks and information technology 2011, 319-324

[12] Daneva, M., & Herrmann, A (2008) Requirements prioritization based on benefit and cost prediction: a method classification framework. Press 34th Euromicro conference software engineering and advanced applications, IEEE computer society, 240-247.

[13] Palma, F., Susi, A., Tonella, P (2011) Using an SMT solver for interactive requirements prioritization. Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on foundations of software engineering ACM, 48-58.

[14] Racheva, Z., Daneva, M., Herrmann, A., & Wieringa, R.J. (2010). A conceptual model and process for client-driven agile requirements prioritization. Fourth international conference on research challenges in information science (RCIS), 287-298.

[15] Tonella, P., Susi, A., & Palma, F (2012) Interactive requirements prioritization using a genetic algorithm. Information and software technology 57-66.

[16] Tonella, P., Susi, A., Palma, F (2013) Interactive requirements prioritization using a genetic algorithm. Inf Softw Technol 55:173-187.

[17] Tao Lu, Ting Li, "Predicting Customer's Preference in E-Commerce Recommendation System: A Genetic Algorithm Approach," icicic, pp.420, Second International Conference on Innovative Computing, Information and Control (ICICIC 2007), 2007.

[18] Xiaoyuan Zhang, Jianzhong Zhou, Changqin Wang, etc. (2012) Multi-class support vector machine optimized by inter-cluster distance and self-adaptive deferential evolution[J], Applied Mathematics and Computation 218(9): 4973-4987.

[19] Jason McZara & Shahryar Sarkani & Thomas Holzer & Timothy Eveleigh, Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment, Empir Software Eng (2015) 20:1721–1761 DOI 10.1007/s10664-014-9334-8.

[20] Seyed Abbas Sadatsakkak, Mohammad H. Ahmadi, Roham Bayat, Seyed Mohsen Pourkiaei, Michel Feidt , Optimization density power and thermal efficiency of an endoreversible Braysson cycle by using non-dominated sorting genetic algorithm, Energy Conversion and Management 93 (2015) 31-39.

[21] S.-Y.Chung, C.Berenstein: !-Harmonic functions and inverse conductivity problems on networks. SIAM J. Appl. Math., 65 (2005), 1200–1226.

[22] Sever, A. (2011) A problem of Data Mining in E-Commerce, Applied Mathematics and Computation, Volume 217, Issue 24, Pages 9966 – 9980

[23] Svahnberg M, Goreschek T, Feldt R, Torkar R, Saleem SB, Shafique MU (2010) A systematic review on strategic release planning models. Inf Softw Technol 52:237-248.

[24] J. del Sagrado, I.M. del Aguila, F.J. Orellana, Multi-Objective ant colony optimization for requirements selection, Journal of Empirical Software Engineering (2014) doi: 10.1007/s10664-013-9287-3.

[25] Svensson, R.B., Goreschek, T., Regnell, B., Torkar, R., Shahrokni, A., Feldt, R., & Aurum, A (2011) Prioritization of quality requirements: state of practice in eleven companies. Proceedings of the requirement engineering conference (RE), 2011 19th IEEE International 67-78.

[26] A. Charan Kumari, K. Srinivas, M.P. Gupta, Software Requirements Optimization Using Multi-Objective Quantum-Inspired Hybrid Differential Evolution, in: Schuze, O., et al. (Eds.), EVOLVE – A Bridge between Probability, Springer, New York, 2013, pp. 107-120.

[27] Tonella, P., Susi, A., & Palma, F (2010) Using Interactive GA for requirements prioritization. 2nd international symposium on search based software engineering, IEEE Computer society, 57-66.

[28] J.T. Souza, C.L Brito Maia, T.N. Ferreira, R.A. Ferreira do Carmo, M.M. Albuquerque Brazil, An ant colony optimization approach to the software release planning with dependent requirements, in: Proc. Of the 3th Int. Symposium on Search Based Software Engineering (SBSE '11), 2011, pp. 142-157.

[29] Cai, L. & Zhu, Y., (2015). The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. Data Science Journal. 14, p.2. DOI: <http://doi.org/10.5334/dsj-2015-002>