Classification of Software Requirements Priorities

[1]Bruno Rossi, [2]Nivir Kanti Singha Roy

[1]Department of Computer Systems and Communications
Masaryk University, Brno, Czech Republic
brossi@mail.muni.cz
[2]Göteborg University & System Engineer at Ericsson AB
Göteborg, Sweden
nivir.roy@gmail.com

Abstract
Requirements Engineering deals with the identification, elaboration, tracking of requirements specifications that drive the software development process. In this area, one key decision is about the requirements that should be developed first, given that in the majority of the projects it is not possible to comply with all the requirements that are set. For this reason, the prioritization of requirements deals with the definition of approaches, methods and techniques to provide prioritization of different requirements. In this sense, key roles are played by stakeholders involved and different criteria used for the prioritization (e.g. value given by stakeholders to implemented requirements, risks of implementation, costs of implementation, and so on).
In the current paper, we deal with the problem of scalability of some requirements prioritization techniques, that is in presence of large set of requirements, the process of prioritization becomes quite costly in terms of effort associated to the prioritization process itself. We propose a classification of requirements based on text mining and machine learning, mimicking what has been done successfully in the area of software repositories by classifying the severity of bug reports. For this reason, we propose a link to our previous work in the area of severity classification, applying similar approach within the field of requirements engineering. For the purpose of validating the approach, we apply it to a dataset of software requirements to evaluate the accuracy of the results.

Abstrakt
Analýza požadavků se zabývá identifikací, zpracováním a sledováním specifikací požadavků, které řídí vývojový proces. Vzhledem k tomu, že ve většině projektů není možné vyhovět všem stanoveným požadavkům, určení těch požadavků, které mají být provedeny jako první, představuje klíčové rozhodnutí v této oblasti. Z toho důvodu, prioritizace požadavků se věnuje definicí přístupů, metod a technik pro stanovení priorit různých požadavků. Klíčovou roli hrají zúčastněné osoby a různá kritéria pro stanovení priority (např. Hodnota přidělena zúčastněnými osobami k jednomu implementovanému požadavku, rizika spojená s implementací, náklady na implementaci atd). V tomto článku se věnujeme problému škálovatelnosti některých technik na prioritizaci požadavků. V případě velkého množství požadavků se proces jejich prioritizace stává poměrně nákladným vzhledem k vynaloženému úsilí na samotnou prioritizaci. Navrhujeme klasifikaci požadavků založenou na dolování dat z textu a strojovém učení, přičemž napodobujeme postupy, které byly úspěšně aplikovány v oblasti klasifikace závažnosti bug reportů v softwarových úložištích. V tomto smyslu navrhujeme propojení s naší předchozí prací v oblasti klasifikace závažnosti bug reportů využitím podobného přístupu v prostředí analýzy požadavků. Tento přístup validujeme jeho aplikací na datový soubor softwarových požadavků kvůli vyhodnocení přesnosti výsledků.

Keywords
Software Requirements Engineering, Software Requirements Prioritization, Data Mining, Natural Language Processing, Classification and Prediction.

Klíčová slova
analýza požadavků softwaru, prioritizace požadavků softwaru, dolování z dat, zpracování přirozeného jazyka, klasifikace a predikce.

1 Introduction

A software requirement represents the unambiguous definition of users needs with respect to the business context of a software system under development. All together, the defined requirements constitute the so-called *requirements specifications* that summarize all the details of implementation of a software system. Independently from the software development process methodology adopted (e.g. *agile* or more *heavy-weight*), every methodology has its own way of defining requirements, tracing them over time, and possibly updating them over time - as the user needs and environmental context change.

Software Requirements Prioritization (RP) is the part of Requirements Engineering (RE) that involves the definition of requirements on a scale of importance. Even though the definition of the concept of requirements priority in RE is not unanimous among authors (see for a discussion, Firesmith [5]), requirements prioritization can be seen as the process in which we start with a set of unambigous and well defined requirements, and end with a ranked list. The way in which the requirements are ranked in order of importance can vary widely.

It is now common knowledge that requirements specifications in commercial projects include typically a large number of requirements, and implementing all them is not simply feasible according to the given constraints. As such, RP approaches are important for the selection of requirements that are more important [6]. For this reason, requirements prioritization is quite important to focus the development on the most relevant requirements.

In one of our previous works, we conducted - by means of a mapping study - research about the status of the area of requirements prioritization [12]. We found a growing research interest in the area with empirical focus on the accuracy of the techniques used for the prioritization process and mostly validated with case studies within industry. In this context, we set some recommendations based on the research review, for example to provide a more broader discussion about the attributes considered on the studies on prioritization, and not just focus on the accuracy of the results from the prioritization process. The current paper goes into this direction by focusing more on the level of the scalability of the prioritization process, that is in presence of large set of requirements, the process of prioritization becomes quite costly in terms of effort associated to the prioritization process itself. We propose the classification of requirements based on text mining and machine learning techniques, mimicking what has been done successfully in the area of software repositories by classifying the severity of bug reports. In this sense, we propose a link to our previous work in the area of severity classification [14], applying similar approach within the field of requirements engineering.

For the purpose of validating the approach, we apply it to a dataset of software requirements to evaluate the accuracy of the results. As we will see, requirements are generally lower in number compared to bugs in issue trackers and this poses some issues in the application of the approach, but the large number of requirements is also a condition for the usefulness of the approach with an order of requirements between 150-200 to start benefiting from the results.

The paper is structured as follows: section 2 reviews software requirements prioritization from the angle that is appropriate for the current paper and provides our problem statement. Section 3 presents the approach with the definition of requirements features used for classification. Section 4 presents the results of the application of the proposed approach to a requirements dataset. Section 5 closes the paper with discussions, conclusions and future works.

2 Techniques for Requirements Prioritization

Berander *et al.* [2] describes different aspects of requirements prioritization, distinguishing among: *a) techniques*, that is structured approaches for the prioritization of requirements, like the Planning Game technique common in agile methodologies, *b) activities*, that is particular tasks performed within a technique, e.g. the ranking algorithm, *c) methods*, broader than a technique, e.g. the application of several techniques in a framework for prioritization, *d) processes*, that is considering the process of prioritization, e.g. how stakeholders are involved. All these concepts are part of the requirements prioritization area, and generally the focus is on one of these not all at the same time. The current paper deals with the techniques for requirements prioritization, so we do not detail the other aspects that would be in any case relevant when considering requirements prioritization from a general point of view.

Prioritization Methods	Measurement methods
Analytical Hierarchy Process (AHP)	Pairwise comparison among all requirements to each other. Comparison formula: n(n-1)/2 (where n is the number of requirements)
Cumulative Voting (100-dollar test)	Imaginary units distributed among all the requirements and sum up the with a fixed predefined value i.e. 100 or 1000 dollar
Numerical Assignment	Each requirement is expressed in numerical assignment which are categorised into three groups namely-critical, standard, and optional
Hierarchical Cumulative Voting (HCV)	From high level abstraction level cumulative voting approach is applied in several levels of requirements
Likert Scale Method	Based upon importance, requirements are classified in bipolar scale i.e. very important, very unimportant and intermediate intervals. Each individual value is divided by summed up value. Formula: $b_i = \sum_{i=1}^{n} a^i / S$
Karl Wiegers	Considering four measured weighted(w) variables namely-benefits(b), disadvantages(d), risk(r), cost(c) requirements are prioritized $P^R = w^b b^R + w^d d^R b / w^c c^R + w^r r^R b$
Kano Model	Requirements are categorised into three level of customer satisfaction: one-dimensional, attractive and must-be requirements.
Binary search tree	Simple ranking, bubble sort and binary search tree
Cost Value Approach	A relative pairwise comparison to determine the cost and the value of the requirement.
Case based Ranking (CBR)	RankBoost, ML technique

Table 1: Some of the main requirements prioritization methods

There is a large number of techniques that can be applied for requirements prioritization. Kukreja *et al.* [8] identified 17 techniques that are the most widely used in industry. In the current overview we just provide a discussion of some of them, evaluating them under the light of scalability concerns. We summarized some of the main techniques in Table 1, together with the major considerations about the application of the techniques from this point of view.

Each of the techniques starts with the goal to reach a ranking of requirements based on the interaction between different interested stakeholders. How this is achieved varies according to the different approaches. To exemplify, some of the approaches require the definition of requirements on an ordinal scale, while others use a ratio scale that provides a finer level of granularity, usually at the expense of a more time-consuming effort.

We detail in the following some of the most use approaches: Analytical Hierarchy Process (AHP), Cumulative voting (CV), Hierarchy cumulative voting (HCV), MoScoW, Ranking and Planning game, then we go into the identification of scalability concerns.

The *Analytic Hierarchy Process* (AHP) is a statistical technique proposed by Thomas Saaty [15] for multi-criteria complex decision-making problems. It can be used to prioritize requirements on the basis of different aspects, like importance, penalty, cost, time, and risk. In this approach, the candidate requirements are compared in pair-wise fashion to determine the extent of how one of the requirements is more important than the other requirement. Stakeholders repeat this process for each of the aspects considered (typically value and cost), and can identify in this way requirements that are more likely to be developed first (e.g. high-value and low-effort requirements. This approach is quite time-consuming and for this reason it is generally run at the beginning of the project, as performing it at every iteration might be unfeasible.

In *Cumulative Voting* - also called *100 Dollar Test* - stakeholders distribute imaginary 100 Dollars among all the requirements according to their preference. This is a more agile approach that takes into account decisions that can be made by stakeholders at each iteration. The prioritization process becomes in this way a sort of game that stakeholders can play iteratively. Also in this case, with the growth of the number of requirements the process can be time-consuming, even though the pair-wise comparison between requirements is done implicitly by stakeholders whereas in other approaches it is formalized (e.g. AHP).

Similar to Cumulative Voting, in the *Ranking* technique requirements are assigned explicit rank in chronological order from *1*.....*n*, based on the priority evaluated by every stakeholder. Then results from all stakeholders are combined (merged) based also on some weights given to different stakeholders. This is one of the simplest approaches to prioritization, but it can be difficult to track all requirements in case of large number, and as well the accuracy is not as good as other approaches, even though authors suggest that is the constant repetition of less accurate approaches at every iteration that makes them effective [9].

There are several variations of Cumulative Voting, for example *Hierarchical Cumulative Voting* starts from the assumption that there are different levels of abstractions in requirements and for this reason it provides the complete decomposition of the upper hierarchical level. So the approach of prioritization deals with different abstraction levels so that comparisons are done at the same abstraction level and can be cumulatively evaluated at the other levels [3].

In the *MoScoW* approach, requirements are categorised into four categories, namely MUST have, SHOULD have, COULD have, and WON'T have. Within the MoScoW approach, the classification of requirements is on an ordinal scale, without indication of degree of differences among all requirements. It can happen in this sense that requirements are in the same category of priority level - that is they are considered at the same level of importance while the application of other techniques may distinguish further. This characteristic can be the main advantage and as well the main drawback of the approach, as defining categories can make the selection of requirements for the next iteration more complex, because discussion of the single requirements need to be undertaken after the prioritization process has been completed.

Planning game is another prioritization technique that is a combination of a numerical assignment technique and ranking techniques mostly used in agile software development. In this approach requirements are divided considering three categories: those without which the system will not function, those that are less essential but provide significant business value and those that would be nice to have. After categorization several aspects such as: cost, time, risk, etc... are considered and sorting is performed in a ordinal scale specifying those that can be estimated precisely, those that can be estimated reasonably well and those that cannot be estimated at all [10].

2.1 (Scalability) Limitations of Current Prioritization Techniques

Given the most known techniques for prioritization, these have still some limitations. For example, an interesting review is in Vestola [17], that conducted a empirical study that shows the open issues and limitations of existing prioritization techniques. In their study authors performed a comparison among nine basic requirement prioritization techniques, and the conclusion of the study was that all the techniques have some limitation in different context of prioritization process, for example related to accuracy of the results or effort requested by the stakeholders.

Another interesting study by Aasem *et al.* [1] showed a comparison among requirements prioritization techniques considering different criteria such as scale, granularity, sophistication, aspect, perspective, type etc. Analysing the applicability of existing techniques on software development they proposed a prioritization framework which combines existing prioritization techniques and hampers some of the limitations of the current techniques.

In this paper, we deal with one limitation of some of the prioritization techniques: scalability of the process once the number of requirements grows. For example, an approach such as AHP has an exponential behaviour given the growth of requirements and stakeholders. To exemplify, AHP requires $n(n-1)/2$ comparisons that need to be performed by stakeholders comparing each pair of requirements [7]. Since all the requirements are compared among each other, this leads to a complexity of $o(n^2)$ that is not manageable when the number of requirements grows over a certain limit.

To reduce scalability of requirements prioritization techniques, our proposal is to consider a classification of requirements based on their priority so that new requirements could be classified according to the past history of the project. Considering a feature vector that represents each requirement we can apply a classifier to determine the priority class of each requirement. Such information could be useful to stakeholders during the prioritization process or even as alternative to the prioritization process one enough requirements have been defined for the training process of our supervised classification algorithms.

In the current requirements prioritization scenario, there is one current work that we can compare to our, in the sense that machine learning has been applied to requirements to reduce the problem of scalability in the prioritization process. Perini et al. [13] proposed the CBRank requirement prioritization technique which combines projects stakeholders preferences with requirements ordering approximations computed through machine learning techniques. Case based reasoning (CBR) is a machine learning technique that has been applied on top of the AHP technique to reduce the scalability issues. Rather than applying rule in each case, CBR compares the newly changed requirements with the most similar case. The approach incorporates the Rank boost algorithm and has been tested on a set of 90 requirements.

We detail now the proposed approach starting with the modelling of requirements according to their priority level and features necessary for the prioritization process.

3 Requirements Definition & Priority Classification

A first step for our classification activity is to determine which features of requirements could be useful to build a feature vector with semantically relevant information for a classification model. It can be useful from a conceptual point of view to see characteristics that can be important.

A software requirement contains generally the following data that can be potentially relevant for a classification process:

- *Title*, the name of the requirement;
- *Description*, the textual description of the requirement;
- *Rationale*, the justification of the requirement to understand why requirement is needed to be implemented;
- *Dependencies*, the requirements that needed to be implemented before/after the current requirement or are in any case related to the current one;
- *Hierarchical structure* of the requirements;

Almost independently from the underlying requirements engineering process, software requirements are represented by a series of features that can be useful for the classification of priorities. One general characteristic is the fact that requirements are generally hierarchical (as shown in Fig. 1) and contain dependencies with other requirements.

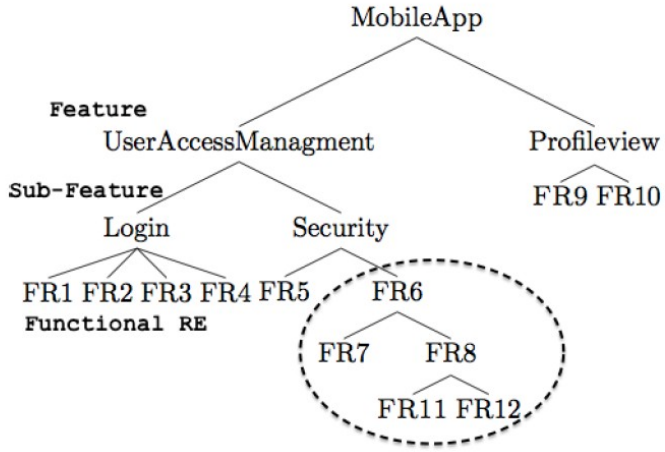


Fig. 1: hierarchical characteristics of functional requirements (FR)

In the context of this paper, we consider the textual description of requirements as our feature vector representing each requirement. In our previous work [14] we proposed a similar approach to improve the classification of severity of issues in bug tracking systems, by considering n-gram models and feature selection. In short, applied to requirements, what we are going to solve is a typical classification problem in which we want to classify new instances of requirements depending on your priority class. To simplify the remaining part of the discussion, we assume that we have a binary classification in which priorities belong to two classes – one higher and one lower – for priorities.

Given a binary priority class for requirements p_0 and p_1, where p_0 is a high priority requirement and p_1 is a low priority one, and given a training set of requirements $r_t = (x; p_i)$ represented by a feature set x and a priority class p_i, we want to classify upcoming requirements (r_e) by means of a classifier f so that:

$$s_i = f_j(r_e),$$

In our case, the classifier *f* is a *Naïve Bayes (NB)* classifier. We decided to use the NB classifier for two reasons, the first one is that it is well-known to perform well for text classification tasks, and the second one is that it can be a baseline for comparison with other more advanced classifiers.

NB is a discriminative model that learns the conditional probability distribution p(yjx), that is the probability of event y given event x. The NB classifier applies Bayesian statistical inference rules commonly used in bag of words models for text classification, e.g. considering (d)ocuments and (c)lasses in our case:

$$P(c|d)=\frac{P(d|c)P(c)}{P(d)}$$

The NB classifier is based upon the assumption that all features are independent to corresponding class label of a given item and the resulting prediction value is based on the presence and absence of a word in a textual document. In order to classify a new item, the NB classifier examines the item features independently and compares those against previous item features. For learning purposes there are two model parameter: the priori probability and the posterior probability or likelihood.

In the prior probability we consider the probability of an event before the evidence is observed, while in the posterior probability or likelihood we compute the conditional probability of each feature value given a certain class. Looking at single features (terms) in an independent way (that is the *"naiveness"* meaning), considering terms x_i:

$$\underset{c\in C}{\operatorname{argmax}}\, P(x_1,x_2,\ldots,x_n|c)\,P(c)=P(x_1|c)\cdot P(x_2|c)\cdot\ldots\cdot P(x_n|c)$$

In this context, we proposed the usage of more advanced language models that uni-grams for modelling of requirements. A language model is a model that represents the probability of words in a text corpora by means of a probability distribution [11]. A uni-gram model considers all the terms in isolation:

$$P(x_1x_2x_3)_u=P(x_1)P(x_2)P(x_3)$$

An example can be seen in the two following sentences: *"one user reported faulty behaviour"* and *"there has been a crash in the user interface"* → *"user"* and *"user interface"* are different concepts for this language model.

An n-gram language model reasons instead on the probability of a word given the context, that is previous *n-1* words, e.g. bi-grams will look at the current and the previous one:

$$P(x_{1,}\ldots.,x_m)=\prod_{i=1}^{m}P(x_1|x_{1,}\ldots.,x_{i-1})\approx P(x_1|x_{i-(n-1)},\ldots.,x_{i-1})$$

This language model allows to give more „semantic" to the represented documents, but we still need to determine which could be the bi-grams that could be more representative. For this, we apply the X^2 (*Chi-Square*) test, based on the assumption that not all the co-locations of terms are relevant and we can generally infer those that are relevant by means of a statistical test. We base this on the assumption of independence, the level of equality of observed frequency of co-locations vs the chance of having single terms:

$$P(x_1x_2)=P(x_1)P(x_2)$$

For all combinations of term we compute the X^2 statistic and then we cut based on a threshold using the terms in the classifier.

3.1 Measures of Accuracy

There are generally several measures of accuracy for a classifier (Table 2) and every measure takes into consideration different aspects of the final performance of the classifier. Generally the most used are accuracy, precision and recall - to this we add the concept of Receiver Operating Curve and the associated AUC measure.

Accuracy shows the overall correctness of the model and is calculated as the sum of correct classifications divided by the total number of classifications. Precision shows the proportion of relevant items which are retrieved, in contrast to recall that presents the fraction of recalled relevant items that are retrieved. The ROC curve shows the performance of classification from the point of view of true positive rates versus false positive ones. Classifiers like NB assign probalities of belonging to a class to each instance, and such information can be used to plot a curve that indicates the performance of the classifier over a ”random-guess“ model (the diagonal of the diagram). If crossfold validation is used, the ROC curve plotted considers the overall set of test data available from all the runs. To summarize information from ROC curves, a common metric is AUC that provides information about the size of the area under the curve. ROC curves are in general preferred over precision and recall as they are independent from the dataset class distribution, so that they can provide more comparable results [4].

	Evaluation focus
$accuracy = \frac{tp+tn}{tp+fn+fp+tn}$	Effectiveness of a classifier in terms of true positives and true negatives
$precision = \frac{tp}{tp+fp}$	Agreement of the classification of positive labels within the dataset
$recall = \frac{tp}{tp+fn}$	Effectiveness in the identification of positive classes
ROC	Receiver Operating Curve, plot of true positive rates versus false positive rates
AUC	Area under the ROC curve

Table. 2: Performance Measures for a classifier

For this reason, in the current paper, we use ROC and AUC to report the results in terms of classification. We next show the process of application of the approach and the experimentation with a set of requirements.

4 Analysis & Results

We applied the approach described in the previous section to a set of requirements. To operationalize the process, we set-up the representation in Fig. 2, in which we extract from the requirements set the requirements descriptions and the priority levels for each requirement. We then pre-process the dataset with stopword removal and stemming, and binarize the class of priorities into p_0 and p_1.

We then apply 10-fold cross validation to train and test the classification algorithm on the dataset. Results of every run are then considered to report ROC curves and other measures of performance of the classification.

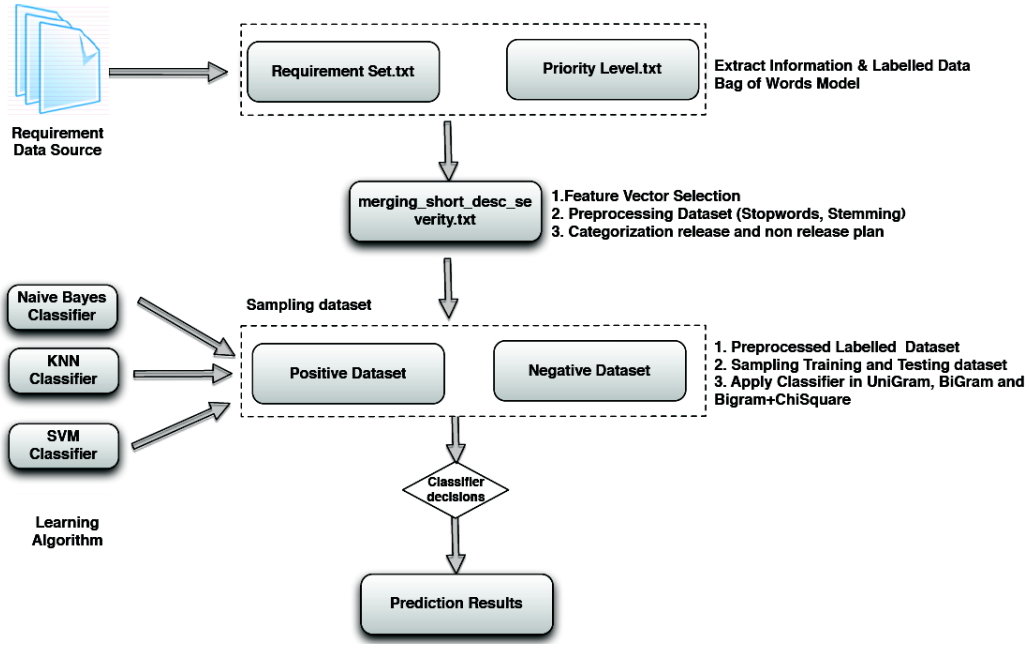


Fig. 2: The process of analysis for the requirements dataset

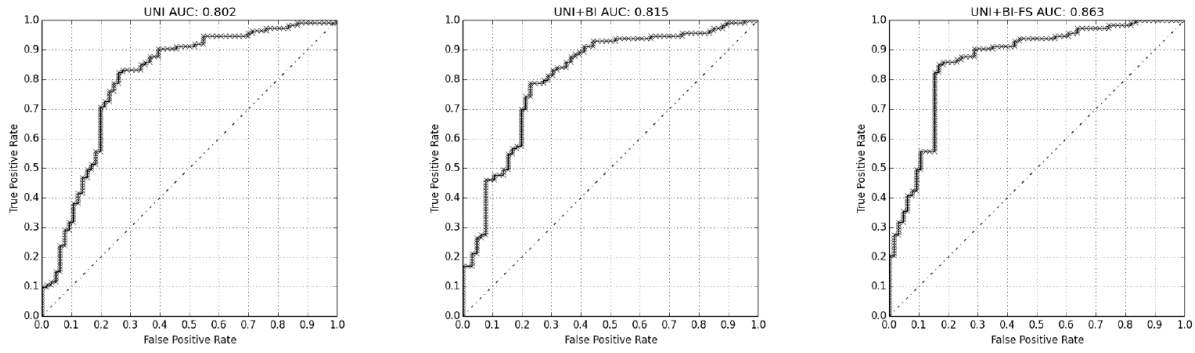
Requirements data-set are considerably smaller compared to bug data-set. Therefore, applying text-mining techniques as with issue trackers can be problematic due to the number of items available for classification. For this reason, we collected a large set of requirements (*179*) that were defined by students during a course about Requirements Engineering. Each requirement contained indication about the priority according to the MoSCoW methodology. Our binarization of the priorities consisted in mapping all the *„Must"* requirements as higher category release (p_0) and other categories as non-release (p_1) that is lower priorities, as in Table 3.

Release			Non-release		
Requirements	**Words**	**Characters**	**Requirements**	**Words**	**Chars**
113	1.032	6.325	66	630	3.726

Table. 3: Requirements dataset descriptive information

We show in Fig. 3 the results from the application of the three models: *Naïve Bayes uni-gram* (*NB-UNI*), *uni-grams+bi-grams* (*NB-UNI+BI*), and *uni-grams+bi-grams+chi-square* (*NB-UNI+BI+CHI*). Results for this specific dataset show small improvement in case of usage of bi-grams, while the improvement is greater when considering feature selection (X^2, *Chi-Square*).

Results show – respectively for the three models - an *AUC* of *0.802*, *0.815* and *0.863*. There is a slight improvement in considering bi-grams for the classification of new requirements, but the largest improvement is given by the consideration of feature selection. The performance of the models can be seen also from the *ROCs* in terms of true positive rates and false positive rates.



(a)Requirement UNI (b)Requirement UNI+BI (b)Requirement UNI+BI+CHI

Fig. 3: Requirements ROC Curves for uni-gram, uni-gram+bi-gram and uni-gram+bi-grams+X^2 feature selection

Feature selection also provides a list of the most relevant terms for the identification of the priority classes. This can be seen as a list of terms that can hint at potentially high (or low) priority requirements. As shown in previous research, such list is domain-specific but also project-specific, so it is unrealistical to use it across different projects, while potentially it can be reused for the same project across time.

Requirement Dataset				
R Term	χ^2		Term	freq
1 event	66.5		form	neg : pos = 14.1 : 1.0
2 form	32.9		icon	neg : pos = 5.1 : 1.0
3 search	25.9		attend	neg : pos = 5.1 : 1.0
4 list	17.3		edit	neg : pos = 4.8 : 1.0
5 view	16.9		list	neg : pos = 4.7 : 1.0
6 edit	16.4		event	neg : pos = 4.7 : 1.0
7 respond	13.1		base	pos : neg = 4.5 : 1.0
8 base	11.3		creat	neg : pos = 4.5 : 1.0
9 contact	10.6		map	neg : pos = 4.4 : 1.0
10 attend	10.5		new	neg : pos = 3.9 : 1.0

Table. 4: Most relevant terms identified by feature selection (chi-square) and pos:neg presence of terms

We report the most relevant terms according to X^2 statistics and frequency in Table 4. The terms ranked in the first 3 columns are those that are more relevant for the classification according to the X^2 statistic. The other two columns report instead terms that are more frequent in one of the two classes (*positive->high priority*, *negative->low priority*). We can see that the terms *"event"* and *"form"* are the two most characterizing terms according to X^2 statistics: for both of them it is about defining negative (low priority) classes. Terms such as *"base"* (and other words based on this root) are instead characterizing for positive (high priority) classes. A full table can contain hundreds of these terms, but the usefulness is more for the classification algorithms than for stakeholders, as interpretation in some cases can be difficult.

5 Conclusions & Future Works

In the current paper, we discussed about the application of classification and text mining to software requirements for their prioritization. The approach can be useful in cases in which there are large

number of requirements and the technique used for prioritization (e.g. AHP) does not scale well with the growth in number of requirements.

We based our approach on our previous research about the classification of severity of bugs in issue trackers ([14]), a promising approach to understand the severity of an issue as it is created – this can give an understanding about how important an issue could be to be solved in the less time as possible (or conversely for non-severe bugs, issues that could be deferred to later stages of the bug resolution process). There are many analogies between requirements and issues, for example both priority and severity are static assignments made by stakeholders and the real decision about the effective implementation/resolution depends on them. In general, these considerations are then translated according to different criterias to the real ordering of resolution (for bugs) or implementation (for requirements). Nevertheless, such information is important to decide the ordering for each iteration (if an iterative approach is used).

Another interesting consideration is that generally requirements are in minor number than bugs in issue trackers. This leads to the fact that the application of text mining and machine learning in general can be problematic. In our experimenatal part we considered a number of requirements near to 200 that we consider a good boundary, but clearly considering lower number of requirements would be a problem. It has to be noted that in any case for lower number of requirements there will be no added benefit in applying the method, in the sense that the also techniques that do not scale well with the increasing number of requirements can be used effectively in presence of a limited number of requirements.

In the experimentation run with the sample dataset, we proposed our approach based on the *Naïve Bayes* classifier using as feature-set the textual description of requirements modeled as uni-gram or bi-gram models with the application then of feature selection to help in the identification of the most relevant terms for the classification process. This is an approach that we consider as a baseline for further comparisons with more advanced approaches.

There are some future works that we can discuss. The first one is to look into existing freely available repositories of requirements to make the experimental part comparable across different studies. We are currently evaluating the *RALIC* dataset [16] to see if it could be a useful dataset to test the approach on. Replication of the approach on freely available repositories would be important for requirements as well as we have in cases of issue trackers. Unfortunately, in the requirements field requirements are very often proprietary and so the comparability and replicability of the studies is difficult to achieve.

A second area is about the binarization of priorities that we are currently applying. Our approach is based on binary classification, while very often existing classes are multiple. Still, with our approach we can distinguish between higher level and lower level priority classes that very often is all that is needed by stakeholders. We planned however to adapt the method for multi-class classification.

A third area is to look at the modelling of additional features to be considered for the classification process. Information about dependencies, hierarchical structure of requirements could be useful to help the classifier in providing more accurate classes based on this additional information. So the proposed approach will need to be adapted with the modelling of a more advanced feature vector, not only represented by textual representations.

Some less important future works are in the evaluation of different classifiers, but this is one aspect more relevant for classification than for the specific domain of requirements - this is one reason why we did not elaborate extensively the selection of the *Naïve Bayes* classifier in the current paper – clearly there are more advanced models (e.g. *Support Vector Machines*) that could provide improvements for the classification of requirements.

6 References

[1] M. Aasem, M. Ramzan, A. Jaffar. Analysis and optimization of software requirements prioritization techniques. In: Information and Emerging Technologies (ICIET), 2010 International Conference on. pp. 1 –6 (june 2010).

[2] P. Berander and A. Andrews. Requirements prioritization. In A. Aurum and C. Wohlin, editors, Engineering and Managing Software Requirements, pages 69-94. Springer Berlin Heidelberg, 2005.

[3] P. Berander and M. Svahnberg, "Evaluating Two Ways of Calculating Priorities in Requirements Hierarchies - An Experiment on Hierarchical Cumulative Voting," J. Syst. Softw., vol. 82, no. 5, pp. 836–850, May 2009.

[4] T. Fawcett, "An introduction to roc analysis," Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, 2006.

[5] D. Firesmith. Prioritizing requirements. Journal of Object Technology, 3(8):35-48, 2004.

[6] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements. Information and Software Technology, 39(14-15):939-947, 1998.

[7] J. Karlsson, K., Ryan. A cost-value approach for prioritizing requirements. IEEE Softw. 14(5), 67–74 (Sep 1997).

[8] N. Kukreja, S. Payyavula, B. Boehm, and S. Padmanabhuni. Selecting an appropriate framework for value based requirements prioritization a case study. In Requirements Engineering Conference (RE), 2012 20th IEEE International, page (to appear), 24 2012-sept. 2012.

[9] D. Leffingwell. „Agile software requirements: lean requirements practices for teams, programs, and the enterprise". Upper Saddle River, NJ: Addison-Wesley, 2011. xxxv, 518.

[10] D. Leffingwell. Managing software requirements: A use case approach. pp. 2nd edition, pp 124–125. Addison-Wesley (USA, May 2003)

[11] D. Manning. Foundations of statistical natural language processing. Ed. Hinrich Schütze. MIT press, 1999.

[12] M. Pergher, and B. Rossi. Requirements Prioritization in Software Engineering: A Systematic Mapping Study in 2013 IEEE Third International Workshop on Empirical Requirements Engineering (EmpiRE).

[13] A. Perini, A. Susi, P. Avesani. A machine learning approach to software requirements prioritization. Software Engineering, IEEE Transactions on PP(99), 1 (2012).

[14] N. Roy, and B. Rossi. Towards an Improvement of Bug Severity Classification in 40th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2014, Verona, Italy, August 27-29, 2014. IEEE.

[15] T. L. Saaty. "Analytic Hierarchy Process", in Encyclopedia of Biostatistics, John Wiley & Sons, Ltd, 2005.

[16] S. L. Lim and A. Finkelstein. StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. IEEE Transactions on Software Engineering. Issue 3 Volume 38, pages 707 – 735.

[17] M. Vestola. "A Comparison of Nine Basic Techniques for Requirements Prioritization." Helsinki University of Technology (2008).