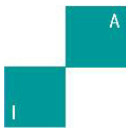
Inteligencia Artificial 44(2009), 35-45
doi: 10.4114/ia.v13i44.1044



INTELIGENCIA ARTIFICIAL

http://erevista.aepia.org/

Semi-Supervised Classification of Non-Functional Requirements: An Empirical Analysis

Agustin Casamayor, Daniela Godoy, Marcelo Campo

ISISTAN, Instituto de Sistemas de Tandil, Facultad de Ciencias Exactas, UNICEN
Campus Universitario, Paraje Arroyo Seco, Tandil (B7001BBO) Bs. As., Argentina
CONICET, Consejo Nacional de Investigaciones Científicas y Técnicas
Ministerio de Ciencia, Tecnología e Innovación Productiva, Presidencia de la Nación - República Argentina

{acasamay,dgodoy,mcampo}@exa.unicen.edu.ar

Abstract The early detection and classification of non-functional requirements (NFRs) is not only a hard and time consuming process, but also crucial in the evaluation of architectural alternatives starting from initial design decisions. In this paper, we propose a recommender system based on a semi-supervised learning approach for assisting analysts in the detection and classification of NFRs from textual requirements descriptions. Classification relies on a reduced number of categorized requirements and takes advantage of the knowledge provided by uncategorized ones as well as certain properties of text. Experimental results show that the proposed recommendation approach based on semi-supervised learning outperforms previous proposals for classifying different types of requirements.

Keywords: non-functional requirements, semi-supervised text learning, requirements classification, recommender systems.

1 Introduction

Requirements management is the process of eliciting, documenting, analyzing, prioritizing and agreeing on requirements between stakeholders. As it is the first step of every software development project, it is a very important and usually time consuming process in which clients and requirement engineers interact in order to agree on what the system has to do. The overall result is a "list" of functional and non-functional requirements (NFRs) that describe the system as a whole and how it should perform, usually without particular details on how it is actually going to do it. NFRs constrain the behavior and development of a software system as they specify overall qualities or attributes the resulting system should have. Examples of NFRs include security, performance, availability, extensibility and portability, among others. These kind of requirements play a critical role in architectural design, so that the early detection of software quality attributes is desirable in order to take them into consideration starting from initial design decisions.

Informal textual descriptions written in natural language are a common means for specifying requirements in early phases of software projects [19]. Natural language requirements let the clients validate that what is written actually represents what they want and they are easy to understand by other stakeholders involved in the project. The study held by Mich et al. [19] showed that most requirements (approximately 79% of the documents) are written in natural language. However, reading, understanding and processing

ISSN: 1988-3064(on-line)

all this text is a very hard and remarkably time consuming task for engineers, which may be lightened by some sort of automatic or semi-automatic computational support.

Numerous attempts have been made to construct automatic tools for assisting developers during the analysis of textual requirements' specifications [8, 11, 15, 22]. Both information retrieval (IR) and natural language processing (NLP) techniques have been applied in the development of tools supporting more efficient automatic or semi-automatic requirement analysis. For textual requirements expressed in natural language, the detection and classification of non functional requirements (NFRs) have been approached using supervised learning techniques [6, 7] and, in some cases, integrated into recommender systems [5, 3].

Recommender systems have become an important research area, mainly due to the abundance of practical applications that help users to deal with information overload by providing personalized recommendations [1]. These systems are characterized by the ability of making recommendations of potentially useful information in many application domains, such as web pages [16], news [23], e-commerce [24] and movies [20], among others. Recommender systems made significant progress over the last decade when numerous content-based, collaborative, and hybrid methods were proposed and subsequently developed [12].

In this paper we introduce a recommender system for the automatic detection and classification of NFRs, using semi-supervised learning and classification techniques [4], that provides assistance to human analysts in an early software development stage. Recommendations made by the system can be used for requirement analysis in software development projects, reducing the effort of manual identification and classification of such documents. In this scenario, requirement analysts should provide a small initial set of categorized documents, obtaining recommendations about a possible classification of the remaining documents. Additionally, analysts can provide feedback to refine classification in an iterative process.

The remaining of this paper is organized as follows. Section 2 presents the proposed approach to a recommender system with semi-supervised categorization of textual requirements for intelligent assistance to requirement analysts. Empirical evaluation of our approach is summarized in Section 3. Section 4 discusses some related work in the use of recommender systems for text analysis in software engineering tasks. Finally, concluding remarks are stated in Section 5.

2 Our Approach

Non-functional requirements are one of the hardest problems analysts and designers have to deal with during software design. NFRs usually specify critical and highly important quality attributes the client asked for his software, and they not only have to be identified within a possible large set of requirement documents, but also classified and prioritized.

In a real-world scenario, the analyst of a software development project needs to go through all the requirement documents gathered by an elicitation team, in order to decide whether they specify functional or non-functional requirements, as well as the categories or classes each NFRs belongs to (i.e. security, availability, scalability, etc.), with the ultimate goal of prioritizing and mapping them into architectural concerns. This is an enormously time consuming task which requires a lot of effort from analysts since every requirement document must be read and manually classified.

In the following subsections, the proposed recommender system that aids analysts in detecting and classifying non-functional requirements is detailed.

2.1 Recommender System Overview

The main goal of the proposed recommender system is providing suggestions to the analyst about candidate NFRs and their corresponding categories. That is to say, given a set of requirement documents written in natural-language, a classifier automatically recognizes whether a given requirement is functional or non-functional, and in the second case, it suggests a suitable category or class (i.e. security, performance, usability, etc.). Afterwards, the recommender system presents the results to the analyst for inspection and correction in case of misclassification, providing valuable feedback for successive iterations.

To accomplish this goal, the system uses some categorized requirements in conjunction with non-categorized ones to learn a text classifier using a semi-supervised learning algorithm. The initial set of categorized requirements can be either detected by the requirement team as they perform interviews with

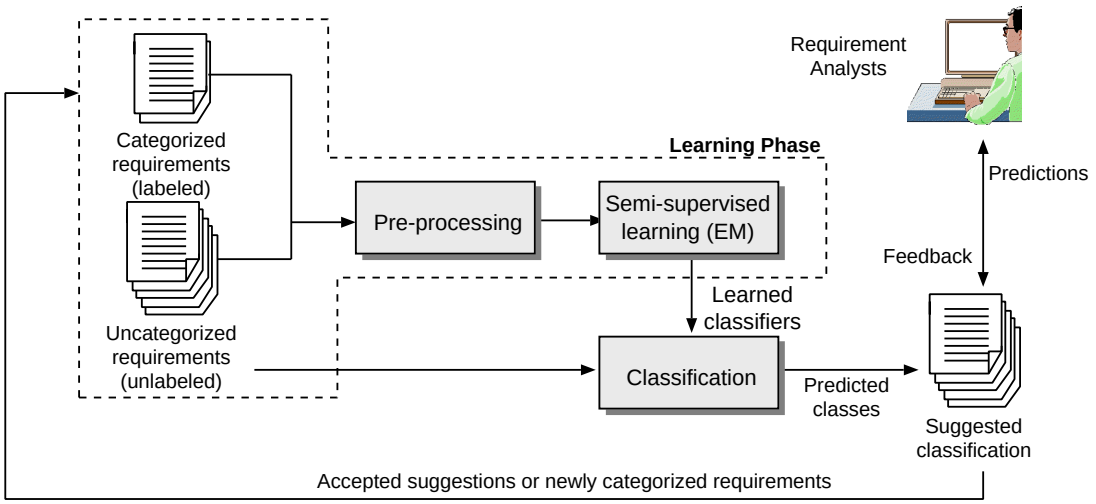


Figure 1: Overview of the recommender system for detection and classification of NFRs

users, or established with some alternative approach (such as the simple method of using a pre-defined fixed set of keywords for classification proposed in [7]). The learned classifier is used to categorize the remaining unlabeled requirements. Optionally, the requirements classified with the highest confidence and/or those which received feedback from the analysts can be used as labeled requirements to repeat the process. Figure 1 depicts an overview of the proposed scheme.

2.2 Semi-supervised Classification of NFRs

The problem of learning classifiers that can predict the class labels of new, previously unseen examples based on a set of labeled training examples is known as supervised learning. Supervised learning involves assigning a set of documents to one or more pre-defined classes or categories $C = \{c_1, c_2, \ldots, c_{|C|}\}$, where each class is supposed to contain documents with certain common properties. In a learning phase, a supervised learning algorithm is applied to induce a classifier, model or hypothesis, which is in turn used to predict the class of new documents in a classification phase.

Naïve Bayes classifier is one of the most popular techniques for text classification and has been reported as performing extremely well in practice in many research studies [18, 10]. The Bayesian approach for classification consists of finding the most probable class for a new example within a finite set of classes given the attributes that describe this example.

The estimate probability of a word w_t given class c_j is the number of times that w_t occurs in the training data D_j divided by the total number of occurrences in the training data for that class. Considering Laplacian smoothing to handle zero counts for infrequent words this can be formulated as follows:

$$P\left(w_t \left| c_j; \hat{\Theta}\right.\right) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N_{ti} P\left(c_j \left| d_i \right.\right)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|\mathcal{D}|} N_{si} P\left(c_j \left| d_i \right.\right)} \tag{1}$$

where N_{ti} is the number of times that the word w_t occurs in the document d_i and $P\left(c_j \left| d_i \right.\right) = 1$ for each document in D_j and $P\left(c_j \left| d_i \right.\right) = 0$ for documents of other classes.

The class prior probabilities can be also calculated using training data as follows:

$$P\left(c_j \left| \hat{\Theta}\right.\right) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} P\left(c_j \left| d_i \right.\right)}{|C| + |\mathcal{D}|} \tag{2}$$

Given estimates of both parameters calculated from training documents, classification can be performed on test documents by calculating the posterior probability of each class given the evidence of the test document, and selecting the class with the highest probability. Using Bayes rule and considering that $w_{d_i,k}$ is the word in position k of the document d_i, this can be formulated as follows:

Algorithm 1 EM algorithm with naÔve Bayesian classification

1: Learn an initial naÔve Bayesian classifier f from only the labeled set L (using Equations 1 and 2);
2: **repeat**
// E-Step
3: **for** each example d_i in U **do**
4: Using the current classifier f to compute $P(c_j|d_i)$ (using Equation 3)
5: **end for**
// M-Step
6: Learn a new naÔve Bayesian classifier f from $L \cup U$ by computing $Pr(c_j)$ and $Pr(w_t|c_j)$ (using Equations 1 and 2)
7: **until** the classifier parameters stabilize
8: Return the classifier f from the last iteration

$$P\left(c_j\,|d_i\,;\hat{\Theta}\right) = \frac{P\left(c_j\left|\hat{\Theta}\right.\right) P\left(d_i\left|c_j;\hat{\Theta}\right.\right)}{P\left(d_i\left|\hat{\Theta}\right.\right)} \approx \tag{3}$$

$$\approx \frac{P\left(c_j\left|\hat{\Theta}\right.\right) \prod_{k=1}^{|d_i|} P\left(w_{d_i,k}\left|c_j;\hat{\Theta}\right.\right)}{\sum_{r=1}^{|C|} P\left(c_r\left|\hat{\Theta}\right.\right) \prod_{k=1}^{|d_i|} P\left(w_{d_i,k}\left|c_r;\hat{\Theta}\right.\right)} \tag{4}$$

It is worth noticing that the transformation from Eq. 3 to Eq. 4 is valid if:

$$P\left(d_i\,|c_j\,;\hat{\Theta}\right) = \prod_{k=1}^{|d_i|} P\left(w_{d_i,k}\left|c_j;\hat{\Theta}\right.\right) \tag{5}$$

which only holds under the assumption of conditional independence.

In supervised learning, the selected algorithm (for example, naÔve Bayes, k-NN, etc.) uses some labeled training examples from every class to generate a classification function or hypothesis. The problem of this approach is the large number of labeled examples required to learn a classifier capable of accurately predicting the label of a novel example. Furthermore, labeling is a time and cost consuming as well as error prone task since it has to be performed manually by domain experts.

Partially-supervised classification implies that there is no need for full supervision, considerably reducing the labeling effort required from users or experts. One of the possible strategies for partial supervision, commonly known as semi-supervised learning, consists of learning from both, labeled and unlabeled examples, or documents in the case of text categorization. This strategy is also known as LU learning (L stands for *labeled* and U for *unlabeled*). LU learning algorithms are based on a small set of labeled examples belonging to each class and a considerably larger set of unlabeled examples that are used to improve learning [17]. Although small, every class must have a set of labeled examples in order to enable learning.

The EM (Expectation-Maximization) strategy [9] fits into a popular class of iterative algorithms for maximum likelihood estimation in problems with incomplete data. It consists of two steps, the *Expectation step* or E-step and the *Maximization step* or M-step. Basically, the first step fills in the missing data based on the current estimation of the parameters and the second step re-estimates the parameters maximizing the likelihood [17]. Unlabeled documents can be regarded as having missing data because of their lack of class labels. The parameters found on the M step are in turn used to begin another E step, and the process is repeated until EM converges to a local minimum when the model parameters stabilize. Nigam et al. [21] proposed the EM algorithm for LU learning with NaÔve Bayes classification, which is summarized in Algorithm 1. The parameters that EM estimates in this case are the probability of each word given a class and the class prior probabilities.

Initially, the documents in the labeled set L have class labels, whereas the documents in the unlabeled set U have missing class labels. EM is used to estimate the missing class labels based on the current

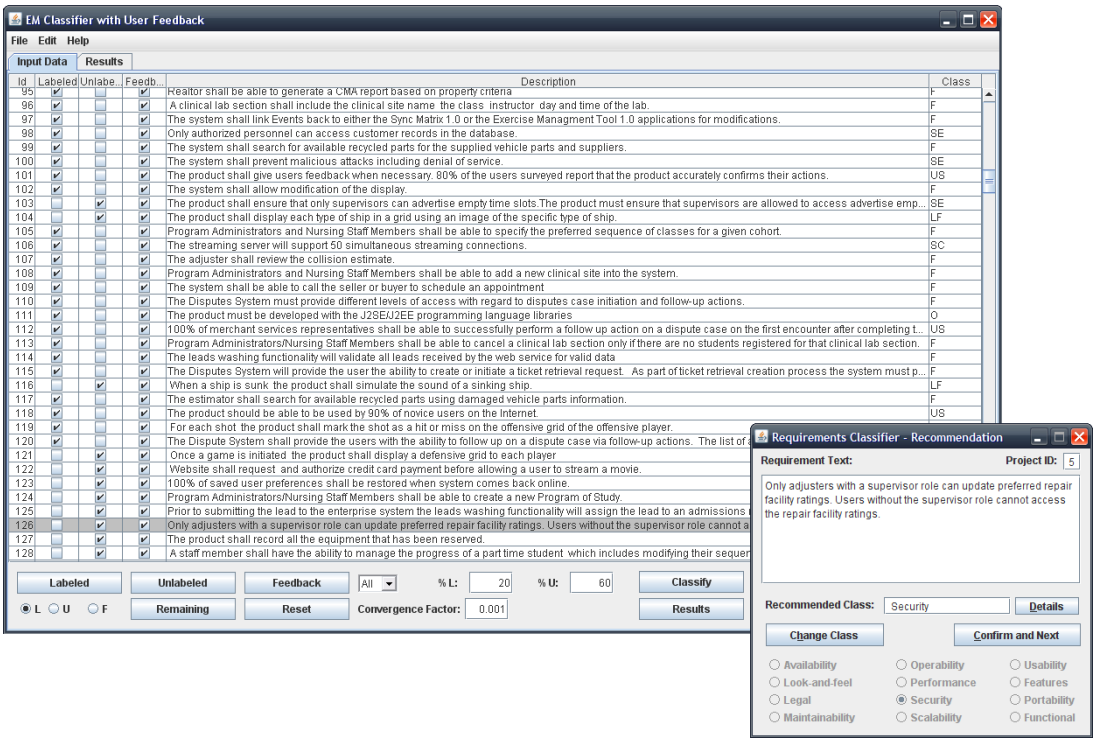


Figure 2: Snapshot of the recommender tool for detection and classification of NFRs

model, i.e. to assign probabilistic class labels to each document d_i belonging to U. Thus, in each iteration EM assign to every document in U a probability distribution on the classes that it may belong to, i.e. $P(c_j|d_i)$ which takes a value in the$[0, 1]$ interval. Instead, documents in L belong to a single class c_k that is known beforehand, i.e. $P(c_k|d_i) = 1$ and $P(c_j|d_i) = 0$ for $j \neq k$. Using both the labeled set L and the unlabeled set U with the assignments of $P(c_j|d_i)$, a new naÔve Bayes classifier is constructed. This gives place to the next iteration of the algorithm, continuing until the classifier parameters, i.e. $P(w_t|c_j)$ and $\mathrm{P}(c_j)$, no longer change or exhibit minimum changes.

The basic EM approach is based on the assumption that there is one-to-one correspondence between mixture components and classes. For textual data, a violation of this assumption is equivalent to saying that a class may consist of several different sub-classes or sub-topics, each best characterized by a different word distribution. In principle, the assumption holds in the case of requirements since NFRs focuses in certain software characteristics instead of multiple topics.

In this work, the EM strategy was implemented as it is described in Algorithm 1 using *Java* with the NaÔve Bayes algorithm provided by the *Classifier4J*[1], a *Java* library designed for text classification. This implementation was used to carry out an experimental evaluation of the approach using a dataset of functional and non-functional requirements. A sample snapshot of the recommender tool with support for user feedback is shown in Figure 2

3 Empirical Evaluation

In our recommendation approach, the identification of NFRs is focused on semi-supervised text classification. Thus, the main aspect to evaluate is the performance of this kind of classifiers in categorizing a set of textual requirements, particularly materialized through the Expectation Maximization strategy. For comparison purposes, we also tested some other common supervised text classification approaches, including the Rocchio algorithm [14], which is also known as *TF-IDF* classifier (bacause this is the

[1] http://classifier4j.sourceforge.net/

most frequently used weighting scheme, along with the cosine similarity measure), *k-NN* [2] and *naÔve Bayes* [25].

NFRs are concerned with different software quality attributes a system must exhibit, such as accuracy, performance, security and modifiability. For the N different non-functional properties to be considered, N binary classifiers are learned, each one trained to distinguish the requirements belonging to a single class from those in all of the remaining classes. Hence, during the training process, a document that belongs to a class is added as a positive example to the corresponding classifier and as a negative example to the rest of the binary classifiers. At classification time, every document is tested against the N classifiers and the classifier with the highest output function or probability score assigns the class label. This is also referred to as *one-vs-all* scheme in multi-class binary classification.

We used a collection of documents expressing a number of requirements available at PROMISE Software Engineering repository[2] in order to perform the experiments. This dataset was built by MS. students at DePaul University during a graduate course on Requirements Engineering. The collection consists of a total of 370 NFRs and 255 functional ones, corresponding to 15 different software development projects. Each document is composed by a description of the requirement, written in natural language, the ID of the project to which it belongs to and a label specifying the type of requirement. These labels indicate either that the requirement is a functional one or the type of non-functional requirement including the quality attributes availability, look-and-feel, legal, maintainability, operational, performance, scalability, security, usability, features and portability. NFRs are divided into the mentioned 11 different classes. However, the quality attribute portability had a single example in the collection and was excluded from the experiments due to its low incidence.

The purpose of requirement classification is to identify the type of each textual requirement in the classes described above. The results of this classification process were evaluated using a multi-label adaptation for the standard definitions of accuracy, precision and recall metrics [26]. This adaptation consists of computing an aggregate score across categories, averaging the scores of all binary tasks. These resulting scores are called macro-averaged accuracy, precision and recall, respectively.

For each experiment, we randomly split the collection into a training set, which is used to learn binary classifiers for requirements, and a testing set, which is used to evaluate their joint performance in classifying previously unseen requirements. Every experiment was ran 10 times using stratified 10-fold cross-validation in order to obtain average scores for the metrics mentioned above. Since the collection used for experiments has an unbalanced distribution of examples, stratification is used to ensure that each fold contains roughly the same proportion of examples in each class as in the original collection. It is important to remark that every training set needs to have at least one document of each class; otherwise, neither supervised nor semi-supervised classifiers can learn to distinguish examples in that class.

The Expectation Maximization strategy with naÔve Bayesian classifiers was implemented according to the algorithm detailed in Algorithm 1. To evaluate the effectiveness of the classifiers learned with this algorithm, we calculated the accuracy of categorizing the collection of requirements using different sizes of the training set and, consequently, different proportions of labeled and unlabeled examples for learning classifiers. We split the collection into 468 requirements (approximately 75% of the collection) for training, preserving the remaining 156 (approximately 25%) for testing.

Figure 3 compares the experimental results obtained with EM strategy and other classical algorithms for supervised text categorization as naÔve Bayes, k-NN and vectorial word matching with TF-IDF, using the same split of requirements. EM outperforms these algorithms in terms of accuracy since unlabeled requirements provide some insights about the different types of requirements that are exploited during learning, for instance words that tend to appear together in positive examples of a certain type of requirements or belong to negative examples of such type.

These experiments were performed using increasing percentages of the training set as labeled examples and the remaining requirements, also in the training set, as unlabeled ones. The 100% of labeled examples corresponds to the total of the 468 requirements in the training set, whereas the size of the test set is maintained along all the experiments (156 requirements). It can be observed in the figure that the semi-supervised approach proposed in this paper takes advantage of unlabeled examples to improve classification accuracy in comparison with a supervised approach using naÔve Bayes, which is based exclusively on labeled examples.

[2] http://promisedata.org/?p=38

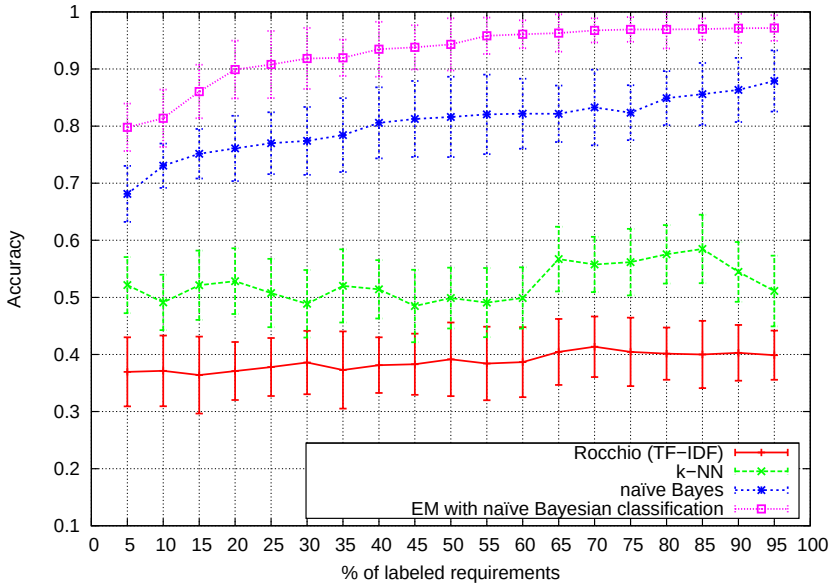


Figure 3: Comparison of several classification algorithms versus semi-supervised EM strategy

In the results described before, accuracy was calculated considering both functional and non-functional requirements of different types (availability, look-and-feel, maintainability, etc.). However, the goal of the classification approach introduced in this paper is to identify NFRs in their corresponding categories. Figure 4 shows the precision of classification for functional requirements and the average for the different categories of NFRs. Naturally, precision is slightly better for functional requirements as they outnumber NFRs in the different categories (255 requirements are functional, whereas NFRs range from 10 to 67 requirements per category).

NFRs are not only distinguished from functional requirements, but also classified into different classes according to their type with a high level of precision. Even using a reduced number of labeled examples for training the classifiers, semi-supervised classification reaches good levels of performance. Figure 4, for instance, shows that EM overcomes the 75% of precision for NFRs with less of 25% out of the total number of training requirements being labeled.

Figures 5(a) and 5(b) detail the classification results for each class of NFRs in terms of precision and recall respectively. Among the categories with the poorest performance are *Features* and *Legal* requirements which are also the ones with the smaller number of examples. Some scores were affected by an initial low recall caused by the variety of candidate classes and the existence of some categories with only a few examples. The effectiveness of all the classifiers improves as more labeled requirements become available and classifiers are able to better distinguish requirements in each class. In a real-world scenario, the increase in the amount of labeled examples will be given by processing feedback from analysts. It is also important to notice that the present collection covers requirements in 15 different projects, so that it may be some variability in the vocabulary employed to describe them. An improvement in classification performance can be expected if documents belonging to a single software project are considered during learning.

4 Related Work

Recommender systems have become popular to assist developers in finding reusable and related content in software engineering tasks. In a recent survey on recommendation systems for software development, Happel and Maalej [12] introduced a novel landscape of software development recommendation systems and lined out several scenarios for knowledge sharing and collaboration, aiming at improving context-awareness and particularly addressing information providers. Castro-Herrera et al. [3] proposed a hybrid

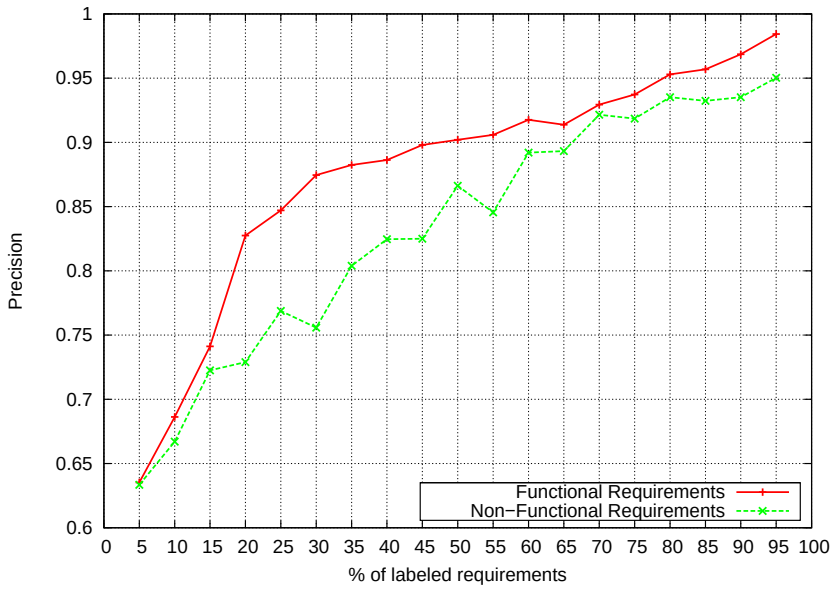


Figure 4: Classification performance for functional and non-functional requirements

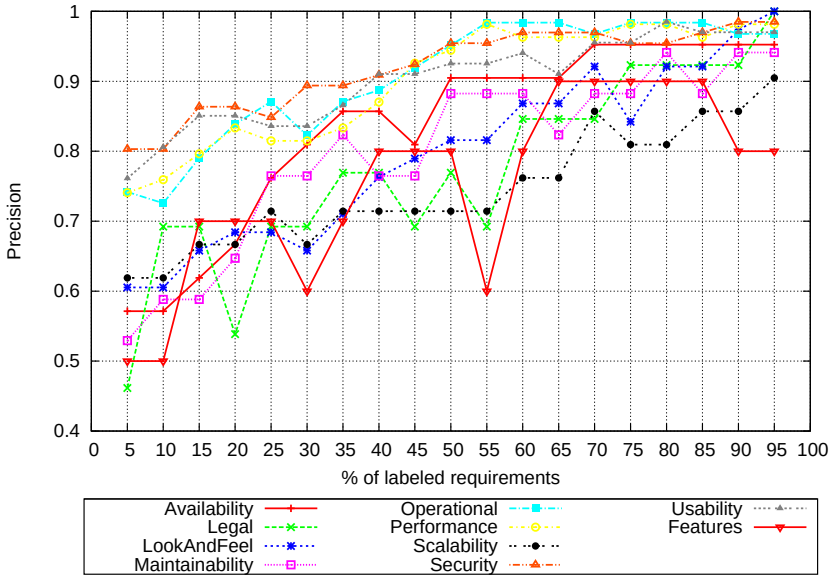
recommender system for requirements elicitation in large-scale software projects, using unsupervised clustering techniques to identify cohesive and finely grained topics, building user profiles according to these topics. However, despite all the advances achieved during the last years, there is still the need for further improvements to make recommendation methods more effective in a broader range of applications [1].

The application of information retrieval (IR) and natural language processing (NLP) techniques for the analysis of requirements expressed in natural-language has been proposed in several works as a method to help analysts in the detection and classification of NFRs concerning different aspects of software, but has not been successfully approached as a recommender system. The problem of detecting and classifying NFRs was tackled from a supervised classification point of view by Cleland-Hung et al. [6, 7]. In these works, a NFR classifier uses a training set of pre-classified requirements to discover a set of weighted indicator terms for each NFR type, i.e. security, performance, etc. Hence, the likelihood of a new requirement to suit a certain NFR type is computed as a function of the occurrence of the corresponding indicator terms within the text of this requirement. The NFR classifier outperforms the results of other supervised classifiers such as naÔve Bayes and standard decision trees, even considering different schemes for feature subset selection [13].

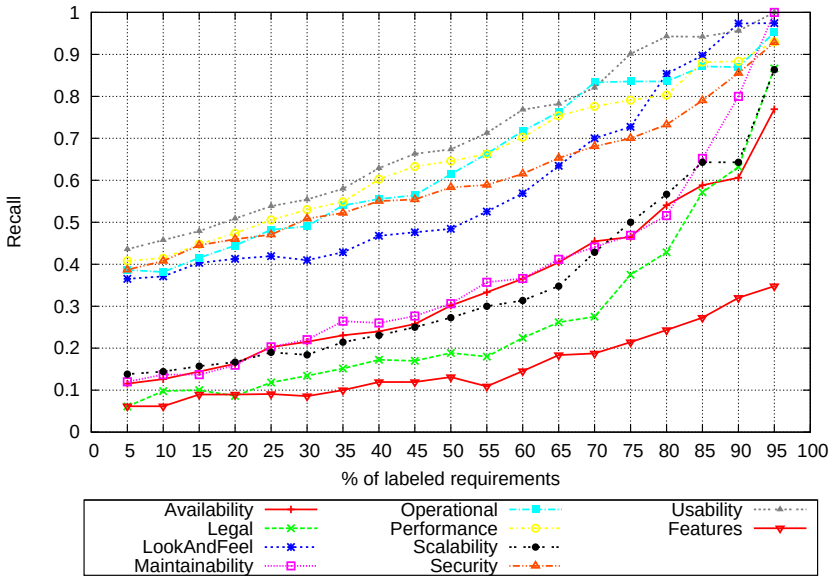
The main drawback of applying supervised methods to NFR detection is related to the amount of pre-categorized requirements needed to reach good levels of precision in the classification process. The NFR classifier uses data from past projects to classify novel requirements in ongoing projects. However, the use of distinctive vocabulary, domain terminology and writing styles across different projects as well as requirement elicitation teams hinder the application of this method. Conversely, the recommendation approach proposed in this paper is based on a classification technique that iteratively classifies requirements gathered for a single project starting from a few categorized requirements and exploiting statistical properties of texts, helping analysts in this hard task.

5 Conclusions

In this paper we introduced a recommender system for detection and classification of NFRs within a set of requirement documents based on semi-supervised learning techniques, aiming at reducing the effort of requirement analysts during an early development stage. Given a small number of manually categorized requirements, the recommender system splits them into functional and non-functional, and then suggests possible categories for the detected NFRs by taking advantage of underlying characteristics of texts.



(a)



(b)

Figure 5: Classification performance for NFRs in terms of precision and recall

Experimental results demonstrate the feasibility of using semi-supervised learning as a method for detecting NFRs. Empirical evidence showed that semi-supervision requires less human effort in labeling requirements than fully supervised methods, and can be further improved based on feedback from analysts within a decision support system for managing requirements as the proposed recommender system.

This approach can help to mitigate the labeling effort required from analysts, involving the manual revision and classification of available textual requirements. In future works we are planning to introduce active learning in this iterative classification process striving to reduce even more the required labeling effort while retaining the accuracy by selecting the examples to be labeled by analysts in an intelligent way (i.e. analysts should be asked to label the more informative examples instead of the top ranked ones).

References

[1] G. Adomavicius and E. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005. doi: 10.1109/TKDE.2005.99.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1st edition, May 1999. doi: 10.1.1.27.7690.

[3] C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1419–1426, New York, NY, USA, 2009. ACM. doi: 10.1145/1529282.1529601.

[4] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*. The MIT Press, September 2006.

[5] J. Cleland-Huang and B. Mobasher. Using data mining and recommender systems to scale up the requirements process. In *Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS'2008)*, pages 3–6, 2008. doi: 10.1145/1370700.1370702.

[6] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 36–45, 2006. doi: 10.1109/RE.2006.65.

[7] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, 2007. doi: 10.1007/s00766-007-0045-1.

[8] J. L. Cybulski and K. Reed. Computer-assisted analysis and refinement of informal software requirements documents. In *Proceedings of the 5th Asia Pacific Software Engineering Conference (APSEC'98)*, page 128, 1998. doi: 10.1145/1370700.1370702.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977. doi: 10.2307/2984875.

[10] S. Eyheramendy, D. D. Lewis, and D. Madigan. On the naive bayes model for text categorization. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 2002. doi: 10.1.1.20/4949.

[11] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of linguistic techniques for use case analysis. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE'02)*, pages 157–164, 2002. doi: 10.1109/ICRE.2002.1048518.

[12] H. Happel and W. Maalej. Potentials and challenges of recommendation systems for software development. In *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, pages 11–15, New York, NY, USA, 2008. ACM. doi: 10.1145/1454247.1454251.

[13] A. Jalaji, R. Goff, M. Jackson, N. Jones, and T. Menzies. Making sense of text: identifying non functional requirements early. Technical report, West Virginia University CSEE, 2006.

[14] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[15] Y. Ko, S. Park, J. Seo, and S. Choi. Using classification techniques for informal requirements in the requirements analysis-supporting system. *Information & Software Technology*, 49(11-12):1128–1140, 2007.

[16] H. Lieberman, C. Fry, and L. Weitzman. Exploring the web with reconnaissance agents. *Commun. ACM*, 44(8):69–75, 2001. doi: 10.1145/381641.381661.

[17] B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer-Verlag, 2007. doi: 10.1145/1540276.1540281.

[18] A. Mccallum and K. Nigam. A comparison of event models for Naïve Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, volume 752, pages 41–48, 1998.

[19] L. Mich, M. Franch, and P. Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004. doi: 10.1007/s00766-003-0179-8.

[20] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl. Movielens unplugged: experiences with an occasionally connected recommender system. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266, New York, NY, USA, 2003. ACM. doi: 10.1145/604045.604094.

[21] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39:103–134, 2000.

[22] S. Park, H. Kim, Y. Ko, and J. Seo. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. *Information & Software Technology*, 42(6):429–438, 2000. doi: 10.1016/S0950-5849(99)00102-0.

[23] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA, 1994. ACM. doi: 10.1145/192844.192905.

[24] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1-2):115–153, 2001. doi: 10.1023/A:1009804230409.

[25] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002. doi: 10.1145/505282.505283.

[26] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1-2):69–90, 1999. doi: 10.1023/A:1009982220290.