Classification of Functional and Non-Functional Requirements using ML Approaches

Sonali Dipak Idate-Mali, Nikhil Soni, Rahul Ratan, Gayan Raja, Saarthak Seth, Pushpendra Kumar Sahu

Abstract— In this paper we take up, the correct classification of requirements has become an essential task within engineering. This study shows how accurately we can automagically classify requirements as functional (FR) and non-functional (NFR) using ML approach in the data set with supervised Machine Learning. There is evidence that ML-based approaches to this problem produce better results than traditional natural language processing (NLP) approaches. Yet, a systematic understanding of these ML approaches is still lacking. The classification algorithms used were Logistic Regression (LR), Support Vector Machine (SVM), Multinomial Naive Bayes (MNB). The use of ML in RE opens up exciting opportunities to develop novel expert and intelligent systems to support RE tasks and processes. Our study discusses the most discriminating features of FRs and NFRs, as well as the sampling strategies used with an additional dataset and their impact on classification accuracy. As future work we intend to compare more algorithms and new forms to improve the precision of our models.

Index Terms— Machine Learning, Support Vector Machines, Imbalanced Data, Classification, Functional Requirements

I. INTRODUCTION

In requirements engineering classifying the requirements of a system by their kind of functional (FR) and non-functional (NFRs). While there is a broad consensus of the definition of FRs, this is rather not the case for NFRs. A Functional Requirements (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behaviour, and outputs while Non-functional requirements (NFRs) define constraints which affects how the system should do it. Functional requirements explain how the system must work, while non-functional explain how the system should perform. The automatic extraction and classification of requirements from text documents has been the focus of several requirements engineering researchers. . The NFRs are system requirements that are orthogonalto the FRs (Yang et al., 2012). NFRs become intertwined with FRs when requirements documents are typically writ- ten and structured around FRs (Cleland-Huang, Settimi, Zou Solc, 2007). To complicate things further, during the design of software architecture, architects need to be able differentiate NFRs from FRs, so that different types of requirements can be transformed into different types of architectural components (Nusseibeh, 2001).

This article presents a systematic review of 24 current ML-based approaches. The review intends to find out what ML algorithms have been used to classify FR and NFRs, how they work and evaluated. In order to improve the state of ML-based approaches, we need to identify what challenges need to be addressed.

RQ1. which features is used for extracting technique works better result for classification into Functional Requirement and Non-Functional Requirement.?

RQ2. Which machine learning Algorithm provides best performance as compare to other for the requirements classification task.?

The main contribution of the work was: Comparison between three important vectorization techniques, the comparison of the use of each one of these techniques combined with four supervised classification algorithms – Random Forest (RF), Decision Trees (DT), Logistic Regression (LR), Support vector Machines (SVM). In this paper, we investigate how automated classification algorithms for requirements can be improved and some of the commonly used machine learning approaches work well in this context. The paper was studied how accurately automatically classify requirements as FRs and NFRs using supervised machine learning. With Support Vector Machine and using lexical features we reach a recall and precision on of 92 for both classes. Furthermore, we assess NFR binary and multiclass classifier for identifying usability, security, operational, and performance NFRs.

II. RELATED WORK

That review included 36 studies that were published between January 1992 and March 2012. Those studies have been analysed by adopting two perspectives: design, which focuses on the technical concepts adopted in each tool, and evaluation, which describes methods for evaluating the effectiveness of those tools From the design perspective, the studies involved in that re- view were classified into four categories: identifying both FRs and NFRs, generating models, analysing quality requirements by defining defects and finding key abstractions. In addition, that review examined the degree of automation for each tool (full or semi- automation) as well as the approaches used to generate knowledge for the reuse of either requirements or knowledge related to requirements. Conversely, the evaluation perspective contains evaluation approaches, concepts, and measures that are used to evaluate tool performance. Identifying requirements automatically is a small part of that review, and these requirements might be either NFRs or FRs, regardless of the techniques used. By contrast, this review focuses only on identifying NFRs via ML algorithms. Kur Tanovic et al., used the Support Vector Machines (SVM) algorithm to classify requirements into Functional Requirements (FR), Non-Functional Requirements (NFR), and the subcategories of non-functional requirements. Researchers used the PROMISE repository, which has the characteristic of being unbalanced in terms of functional and non-functional requirements.

Functional Requirements: According to Pohl et al. is that: Functional requirements require a result of behaviours be provided by a system function. According to IEEE et al., FRs are requirements that specifies a function that a system or system component must be able to perform. Fernandes et al. it is said that functional requirements describe a function to be provided to the users of the system, describing partially its behaviour as a response to the stimulus being applied to it. Technical requirements should not mention any technological issues, that is, functional requirements should be independent of design or implementation aspects.

ISO 16175-1:2020

This document is intended primarily for the purpose of the application, or primarily for enabling other business functions and processes. Implementation guide for applications that manage analogy and / or digital recordings. It is still under development process.

Non-Functional Requirements: The non-functional requirements define the qualities of the system to be developed and often influence the system architecture more than functional requirements. According to Fernandes et al., a non-functional requirement is a set of restrictions imposed upon the system being developed, determining, for example, whether it is attractive, useful, fast, or reliable. Anton et al. defined A NFR is something that describes the non-behavioural aspects of a system. It captures the properties and constraints under which that system must operate, and, according to Davis et al., An overall set of attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, modifiability. etc. NFRs can be divided into subcategories Non-functional requirements are sometimes categorized as product requirements, organizational or process-related requirements, and external requirements.

ISO/IEC 9126

The basic goal of the ISO / IEC 9126 standard is to address well-known human prejudices that can affect the execution and awareness of software development projects. These biases include changing priorities after the project starts and lacking a clear definition of "success." By clarifying and agreeing on project priorities and converting abstract priorities (compliance) to measurable values (output data can be validated against Schema X without intervention), ISO / IEC 9126 seeks to develop a common understanding of project goals and objectives

III. CONTEXT AND DATA SET

The challenge put forward by the Data track of Requirement Engineering consists of taking a given data set and executing an automated Requirement Engineering task on the data, such as tracing, identifying/classifying requirements, or extracting knowledge.

An overview of the data is containing the software requirement feature which is works in the Functional Requirement and Non-Functional Requirement. the labelled is based on Functional Requirement and Non-Functional Requirement '0 & 1' format.

The main data set was balanced by including user comments made on Amazon products. The authors had a precision and recall up to 90 for automatically identifying FRs and NFRs and for the identification of specific NFRs. The reference of data is Kaggle "https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset"

IV. METHODOLOGY

The work is done by four phases to perform the classification Functional Requirement and Non-Functional Requirement.

First step is Normalization, where the data is ready to cleaned. All unwanted words and NULL value is removed. Second step is vectorization, where the data is converted in the format 0 & 1. Third step is classification, where we applied the algorithm, train & predict the classification model of the four algorithms i.e., RF, DT, LR, SVM. Four Step is Evaluation, this is final phase where the result is predicted and the compare all of these and calculate the accuracy.

We have strategies for dealing with imbalances in data, as balanced distribution can improve the classification accuracy. We do random under sampling to achieve a balanced class distribution. The data is converted into 0 & 1 format. '0' denoted by non-Functional and '1' denoted by Functional and then applying the algorithm.

TO ANSWER RQ1. firstly, we convert the data into 0 & 1 because we perform the classification technique. so, we need the data in 'true & false' or '0 & 1' form using the 'pandas. Series' library. All the NULL value is fix.

RQ2. we used three algorithms i.e., Random Forest, Decision Trees, Logistic Regression, Support vector Machines. firstly we take same sample on these algorithm and as compares all of it, so the better result is shows. After that we compares the results obtained by RF, DT, LR, SVM, through performance measures.

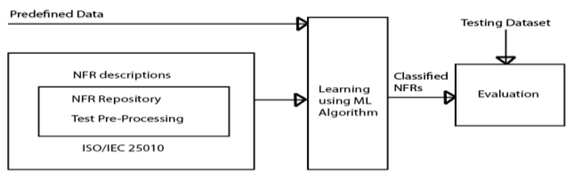


Fig1. A general process of applying ML algorithms to classify NFRs in textual requirements documents.

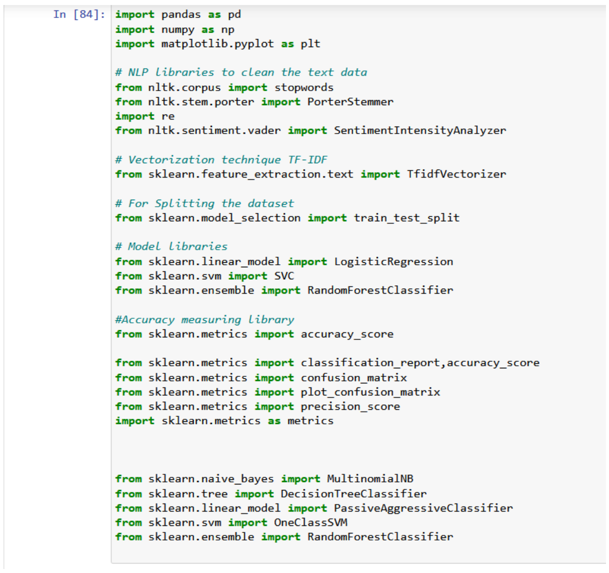
V. RESULT

This section presents the review results and answer our research questions:

RQ2. Which machine learning Algorithm provides best performance as compare to other for the requirements classification task.?

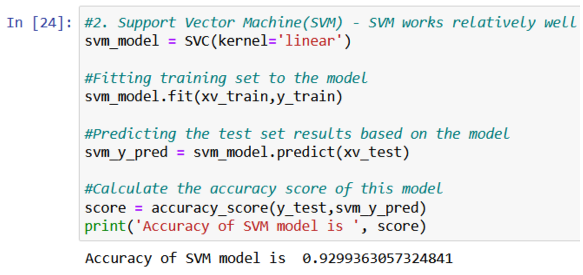
A total of 16 different ML algorithms were found in the 24 selected studies. These algorithms fall into three types, comprising 7 supervised learning (SL), 4 unsupervised learning (USL) and 5 semi supervised learning (SSL). The SL is the most popular type of ML, as it was used in 17 studies (71%), and SVM is overall the most popular ML algorithms, found in 11 studies (45.8%).

Libraries:



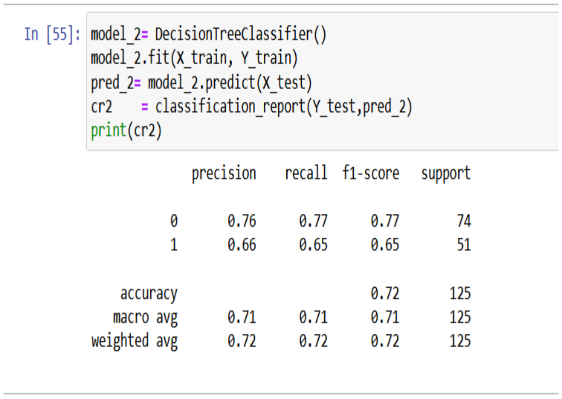
```
In [84]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt

         # NLP libraries to clean the text data
         from nltk.corpus import stopwords
         from nltk.stem.porter import PorterStemmer
         import re
         from nltk.sentiment.vader import SentimentIntensityAnalyzer

         # Vectorization technique TF-IDF
         from sklearn.feature_extraction.text import TfidfVectorizer

         # For Splitting the dataset
         from sklearn.model_selection import train_test_split

         # Model libraries
         from sklearn.linear_model import LogisticRegression
         from sklearn.svm import SVC
         from sklearn.ensemble import RandomForestClassifier

         #Accuracy measuring library
         from sklearn.metrics import accuracy_score

         from sklearn.metrics import classification_report,accuracy_score
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import plot_confusion_matrix
         from sklearn.metrics import precision_score
         import sklearn.metrics as metrics


         from sklearn.naive_bayes import MultinomialNB
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.linear_model import PassiveAggressiveClassifier
         from sklearn.svm import OneClassSVM
         from sklearn.ensemble import RandomForestClassifier
```

Support Vector Machine (SVMs): - It is a group of supervised learning algorithms that can be used for classification and regression. The SVM al go I t hm is based upon the statistical learning theory and the Vapnik–Chervonenkis (VC) dimensions (Vapnik Chervo- nenkis, 2015). The algorithms look for data points that are close to the opposing class. The support vectors are important in the classification task, while the rest of the training data points are irrelevant. In the next step, the best separation line, known as the decision boundary, is found. In S5, a line segregates classes by using different functions, such as the Linear Kernel function where the authors claimed that the linear function is superior to non-linear kernels for classifying textual content (Bennett Bredensteiner, 2000).



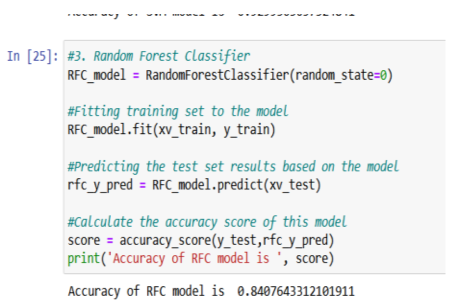
```
In [24]: #2. Support Vector Machine(SVM) - SVM works relatively well
         svm_model = SVC(kernel='linear')

         #Fitting training set to the model
         svm_model.fit(xv_train,y_train)

         #Predicting the test set results based on the model
         svm_y_pred = svm_model.predict(xv_test)

         #Calculate the accuracy score of this model
         score = accuracy_score(y_test,svm_y_pred)
         print('Accuracy of SVM model is ', score)

         Accuracy of SVM model is  0.9299363057324841
```

DECISSION TRESS (DT): Decision Tree: The Decision Tree is the most powerful and popular tool for classification and prediction. The decision tree is a flow chart-like tree structure, where each internal node specifies a test for the attribute, each branch represents the result of the test, and each leaf node (terminal node) contains a class label.

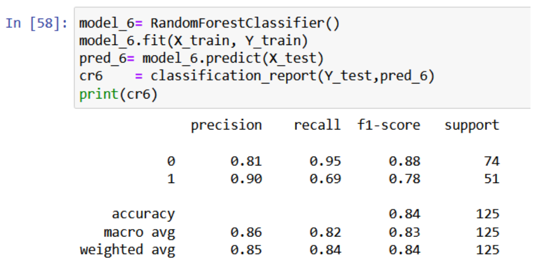


```
In [55]: model_2= DecisionTreeClassifier()
         model_2.fit(X_train, Y_train)
         pred_2= model_2.predict(X_test)
         cr2    = classification_report(Y_test,pred_2)
         print(cr2)

                       precision    recall  f1-score   support

                    0       0.76      0.77      0.77        74
                    1       0.66      0.65      0.65        51

             accuracy                           0.72       125
            macro avg       0.71      0.71      0.71       125
         weighted avg       0.72      0.72      0.72       125
```

Random Forest (RF): The Random Forest classifier creates a set of decision trees from a randomly selected subset of training sets. Basically, it gets a set of decision trees (DTs) from a randomly selected subset of the training set and collects votes from different decision trees to make the final prediction.

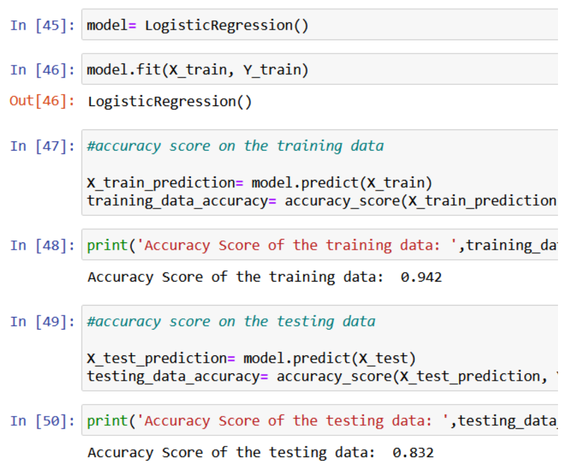


```
In [25]: #3. Random Forest Classifier
         RFC_model = RandomForestClassifier(random_state=0)

         #Fitting training set to the model
         RFC_model.fit(xv_train, y_train)

         #Predicting the test set results based on the model
         rfc_y_pred = RFC_model.predict(xv_test)

         #Calculate the accuracy score of this model
         score = accuracy_score(y_test,rfc_y_pred)
         print('Accuracy of RFC model is ', score)

         Accuracy of RFC model is  0.8407643312101911
```

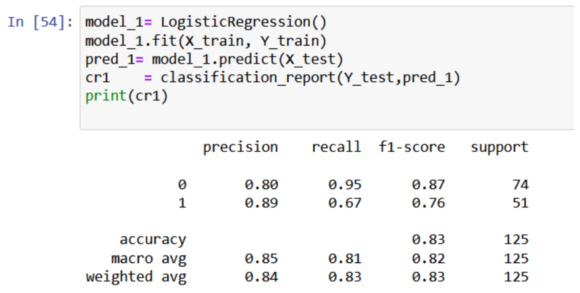


```
In [58]: model_6= RandomForestClassifier()
         model_6.fit(X_train, Y_train)
         pred_6= model_6.predict(X_test)
         cr6    = classification_report(Y_test,pred_6)
         print(cr6)

                       precision    recall  f1-score   support

                    0       0.81      0.95      0.88        74
                    1       0.90      0.69      0.78        51

             accuracy                           0.84       125
            macro avg       0.86      0.82      0.83       125
         weighted avg       0.85      0.84      0.84       125
```

Logistic Regression (LR): Logistic regression estimates the probability that an event will occur. For example, it was selected or not selected based on a particular set of independent variables. The result is a probability, so the dependent variable is limited between 0 and 1. Logistic regression applies logit transformations to the odds. H. The probability of success divided by the probability of failure. This is also commonly referred to as the logarithm or the natural logarithm of the odds.



```
In [45]: model= LogisticRegression()

In [46]: model.fit(X_train, Y_train)
Out[46]: LogisticRegression()

In [47]: #accuracy score on the training data

         X_train_prediction= model.predict(X_train)
         training_data_accuracy= accuracy_score(X_train_prediction

In [48]: print('Accuracy Score of the training data: ',training_da

         Accuracy Score of the training data:  0.942

In [49]: #accuracy score on the testing data

         X_test_prediction= model.predict(X_test)
         testing_data_accuracy= accuracy_score(X_test_prediction, '

In [50]: print('Accuracy Score of the testing data: ',testing_data

         Accuracy Score of the testing data:  0.832
```



```
In [54]: model_1= LogisticRegression()
         model_1.fit(X_train, Y_train)
         pred_1= model_1.predict(X_test)
         cr1    = classification_report(Y_test,pred_1)
         print(cr1)

                       precision    recall  f1-score   support

                    0       0.80      0.95      0.87        74
                    1       0.89      0.67      0.76        51

             accuracy                           0.83       125
            macro avg       0.85      0.81      0.82       125
         weighted avg       0.84      0.83      0.83       125
```

VI. FUTURE WORK Future works,

we aim to: A. Expand Promise to use and explore other feature extraction techniques such as word2vec, AUR-BoW, Hierarchal BoW and Online Unsupervised Multi-view Feature Selection .

B. To avoid redundant information, employ advanced multi-view strategies to combine different feature sets.

C. Analyze the use of other supervised classification algorithms with the use of Neural Networks.

D. Comparing the results of the supervised classification with an unsupervised classification.

E. Finding ways to mitigate the unbalance of the base and improving classification with little training data would be beneficial.

VII. REFERENCE

1) L. Breiman. Random forests. Mach. Learn., 45(1):5–32, Oct. 2001.:

2) R. Caruana. Learning from imbalanced data: Rank metrics and extra tasks. In Proc. Am. Assoc. for Artificial Intelligence (AAAI) Conf, 2000.:

3) A. Casamayor, D. Godoy, and M. Campo. Identifi- cation of nonfunctional requirements in textual specifica- tions: A semi-supervised learning approach. Information and Software Technology, 52(4), 2010:
put. Stat. Data Anal., 38(4):367–378, Feb. 2002.:

4) J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing reg- ulatory codes to product specific requirements. In Proc. of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010.:

5) J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In Requirements Engi- neering, 14th IEEE International Conference. IEEE, 2006.:

6) A. M. Davis. Software requirements: objects, func- tions, and states. Prentice-Hall, Inc., 1993.:

7) J. Eckhardt, A. Vogelsang, and D. M. Fern´andez. Are non-functional requirements really non-functional? an investigation of non-functional requirements in practice. In Proc. of the 38th International Conference on Software Engineering. ACM, 2016.:

8) A. Estabrooks, T. Jo, and N. Japkowicz. A multiple resampling method for learning from imbalanced data sets. Computational intelligence, 20(1), 2004.:

9) W. B. Frakes and R. Baeza-Yates. Information re- trieval: data structures and algorithms. 1992.:

10) Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, Aug. 1997.:

11) J. H. Friedman. Stochastic gradient boosting. Comput. Stat. Data Anal., 38(4):367–378, Feb. 2002.:

12) P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. Machine Learning, 63(1):3–42, 2006.:

13) M. Glinz. On non-functional requirements. In Re- quirements Engineering Conference, 2007. RE'07. 15th IEEE International. IEEE, 2007.:

14) I. Guyon and A. Elisseeff. An introduction to variable and feature selection. The Journal of Machine Learning Research, 3, 2003.:

15) D. J. Hand and K. Yu. Idiot's bayes not so stupid after all? International statistical review, 69(3), 2001.:

16) H. He and E. A. Garcia. Learning from imbalanced data. IEEE Transactions on knowledge and data engineer- ing, 21(9), 2009.:

17) R. C. Holte, L. Acker, B. W. Porter, et al. Concept learning and the problem of small disjuncts. In IJCAI, volume 89. Citeseer, 1989.:

18) C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In Min- ing Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE, 2013.:

19) J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing reg- ulatory codes to product specific requirements. In Proc. of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010.:

20) J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In Requirements Engi- neering, 14th IEEE International Conference. IEEE, 2006.:

21) A. M. Davis. Software requirements: objects, func- tions, and states. Prentice-Hall, Inc., 1993.:

22) J. Eckhardt, A. Vogelsang, and D. M. Fern´andez. Are non-functional requirements really non-functional? an investigation of non-functional requirements in practice. In Proc. of the 38th International Conference on Software Engineering. ACM, 2016.:

23) G. Kotonya and I. Sommerville. Requirements engi- neering: processes and techniques. Wiley Publishing, 1998.:

24) Z. Kurtanovi´c and W. Maalej. Mining user ra- tionale from software reviews. In Proc. of the 25rd IEEE International Requirements Engineering Conference, 2017.: 22) J. Laurikkala. Improving identification of difficult small classes by balancing class distribution. In Conference on Artificial Intelligence in Medicine in Europe. Springer, 2001.:

23) W. Maalej, Z. Kurtanovi´c, H. Nabil, and C. Stanik. On the automatic classification of app reviews. Require- ments Engineering, 2016.:

24) M. P. Marcus, M. A. Marcinkiewicz, and B. San- torini. Building a large annotated corpus of english: The penn treebank. Comput. Linguist., 19(2):313–330, June 1993.:

25) J. R. Quinlan. Induction of decision trees. Machine learning, 1(1), 1986.:

26) C. Schaffer. Selecting a classification method by cross-validation. Machine Learning, 13(1), 1993.:

27) J. Slankas and L. Williams. Automated extraction of non-functional requirements in available documentation. In 1st International Workshop on Natural Language Anal- ysis in Software Engineering. IEEE, 2013.:

28) W. B. Frakes and R. Baeza-Yates. Information re- trieval: data structures and algorithms. 1992.:

29) Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, Aug. 1997.:

30) J. H. Friedman. Stochastic gradient boosting. Com- put. Stat. Data Anal., 38(4):367–378, Feb. 2002.:

31) P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. Machine Learning, 63(1):3–42, 2006.:

32) M. Glinz. On non-functional requirements. In Re- quirements Engineering Conference, 2007. RE'07. 15th IEEE International. IEEE, 2007.:

33) I. Guyon and A. Elisseeff. An introduction to variable and feature selection. The Journal of Machine Learning Research, 3, 2003.:

34) D. J. Hand and K. Yu. Idiot's bayes not so stupid after all? International statistical review, 69(3), 2001.:

35) H. He and E. A. Garcia. Learning from imbalanced data. IEEE Transactions on knowledge and data engineer- ing, 21(9), 2009.:

36) R. C. Holte, L. Acker, B. W. Porter, et al. Concept learning and the problem of small disjuncts. In IJCAI, volume 89. Citeseer, 1989.:

37) C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In Min- ing Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE, 2013.:

38) E. Knauss, D. Damian, G. Poo-Caama~no, and J. Cleland-Huang. Detecting and classifying patterns of requirements clarifications. In 20th IEEE International Re- quirements Engineering Conference. IEEE, 2012.:

39) G. Kotonya and I. Sommerville. Requirements engi- neering: processes and techniques. Wiley Publishing, 1998.:

40) Z. Kurtanovi´c and W. Maalej. Mining user ra- tionale from software reviews. In Proc. of the 25rd IEEE International Requirements Engineering Conference, 2017.:

41) J. Laurikkala. Improving identification of difficult small classes by balancing class distribution. In Conference on Artificial Intelligence in Medicine in Europe. Springer, 2001.:

42) W. Maalej, Z. Kurtanovi´c, H. Nabil, and C. Stanik. On the automatic classification of app reviews. Require- ments Engineering, 2016.:

43) M. P. Marcus, M. A. Marcinkiewicz, and B. San- torini. Building a large annotated corpus of english: The penn treebank. Comput. Linguist., 19(2):313–330, June 1993.:

44) J. R. Quinlan. Induction of decision trees. Machine learning, 1(1), 1986.:

45) C. Schaffer. Selecting a classification method by cross-validation. Machine Learning, 13(1), 1993.:

46) J. Slankas and L. Williams. Automated extraction of non-functional requirements in available documentation. In 1st International Workshop on Natural Language Anal- ysis in Software Engineering. IEEE, 2013.:

47) I. Sommerville and P. Sawyer. Requirements Engi- neering: A Good Practice Guide. John Wiley Sons, Inc., 1st edition, 1997.:

48) L. T. Su. The relevance of recall and precision in user evaluation. Journal of the American Society for Information Science, 45(3), 1994.:

49) R. B. Svensson, T. Gorschek, and B. Regnell. Quality requirements in practice: An interview study in require- ments engineering for embedded systems. In International Working Conference on Requirements Engineering: Foun- dation for Software Quality. Springer, 2009.:

50)

G. Weiss and F. Provost. The effect of class dis- tribution on classifier learning: an empirical study, tech. Technical report, Re ML-TR-44, Department of Computer Science, Rutgers University, 2001.: