AUTOMATED REQUIREMENTS CLASSIFICATION USING FEATURE SELECTION BASED ON LINGUISTIC FEATURES

by

Sercan Çevikol

B.S., Electrical and Electronics Engineering, Boğaziçi University, 1997

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Systems and Control Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Asst. Prof. Fatma Başak Aydemir. After a long break of my study, she accepted me as a thesis student, and during the difficult pandemic times, she always supported me whenever I had a problem or a question about my research or writing. She consistently encouraged me and kept her patience for my prolonged thesis work.

I would also like to thank Dr. Fabiano Dalpiaz, Assistant Professor in the Department of Information and Computing Sciences at Utrecht University in the Netherlands, and Dr. Davide Dell'Anna, a postdoctoral researcher in the Department of Control and Operations of Delft University of Technology in the Netherlands. Without their participation, input and support, this study could not have been successfully conducted.

Finally, I must express my very profound gratitude to my wife, Betül, and my friend Emre Hayretci for providing me with unfailing support and continuous encouragement throughout my study. This accomplishment would not have been possible without them. Thank you.

Sercan Çevikol

ABSTRACT

AUTOMATED REQUIREMENTS CLASSIFICATION USING FEATURE SELECTION BASED ON LINGUISTIC FEATURES

*R*equirements classification is an important problem in organizing the systems and requirements, and it is widely used in handling large requirements data sets. A basic example of a requirements classification problem is the distinction between the functional and non-functional (quality) requirements. The state-of-the-art classifiers are most effective when they use a large set of word features such as text n-grams or part of speech n-grams. However, as the number of features increases, it becomes more difficult to interpret the approach, because many redundant features have to be explored that do not capture the meaning of the requirements. In this study, we propose the use of more general linguistic features, such as dependency types, for the construction of interpretable machine learning classifiers for requirements engineering. Through a feature engineering effort, assisted by tools that interpret graphically how classifiers work, we derive a set of linguistic features. While classifiers that use the proposed features fit the training set slightly worse than those that use high-dimensional feature sets, this approach performs generally better on validation data sets and is more interpretable. We use industry data sets, and we perform experimental runs using several automated feature selection algorithms to explore whether our feature set can be optimized further using one of the automated selection algorithms. Although in some data sets, impressive results were obtained. the automated selection algorithms did not prove a significant improvement, and even, on average, the results were worse than the results we obtained using the set based on linguistic features.

ÖZET

DİL ÖZELLİKLERİNE GÖRE ÖZNİTELİK SEÇİMİ KULLANILARAK YAZILIM İSTERLERİNİN OTOMATİK SINIFLANDIRILMASI

Yazılım isterlerinin sınıflandırılması, sistemlerin ve isterlerin organize edilmesinde önemli bir sorundur ve büyük yazılım isterleri veri setlerinin işlenmesinde yaygın olarak kullanılır. Yazılım isterlerinin sınıflandırma probleminin temel bir örneği, işlevsel ve işlevsel olmayan (kalite) isterler arasındaki ayrımdır. Son nesil sınıflandırıcılar, metin n-gramları veya sözcük türleri n-gramları gibi geniş kelime özellikleri kümesi kullandıklarında en etkilidir. Bununla birlikte, özelliklerin sayısı arttıkça, yaklaşımı yorumlamak daha zor hale gelir çünkü gereksinimlerin anlamını yakalayamayan birçok gereksiz öz niteliğin de araştırılması gerekir. Bu çalışmada, yazılım gereksinim mühendisliği için yorumlanabilir makine öğrenimi sınıflandırıcılarının oluşturulmasında bağlılık türleri gibi daha genel özniteliklerin kullanılmasını öneriyoruz. Sınıflandırıcıların nasıl çalıştığını grafiksel olarak yorumlayan araçlarla desteklenen bir öznitelik mühendisliği ile bir dizi dilsel öznitelik türetiyoruz. Önerilen öznitelikleri kullanan sınıflandırıcılar eğitim setine yüksek boyutlu öznitelik setlerini kullananlara göre biraz daha kötü performans gösterse de, bu yaklaşım genellikle doğrulama veri setlerinde daha iyi performans gösterir ve daha yorumlanabilirdir. Çalışmamızda sektör veri setlerini kullaniyoruz ve öznitelik setimizin daha da optimize edilip edilemeyeceğini keşfetmek için birkaç otomatik öznitelik seçim algoritması kullanarak deneysel çalışmalar gerçekleştiriyoruz. Bazı veri setlerinde etkileyici sonuçlar elde edilmesine rağmen otomatik seçim algoritmaları önemli bir gelişme göstermedi ve hatta ortalama olarak sonuçlar, dil özniteliklerine dayalı seti kullanarak elde ettiğimiz sonuçlardan daha kötüydü.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . x
LIST OF ACRONYMS/ABBREVIATIONS . . . xi
1. INTRODUCTION . . . 1
1.1. Problem Statement . . . 1
1.2. Purpose of the Study . . . 3
1.3. Approach . . . 4
1.4. Organization . . . 5
2. BACKGROUND AND RELATED WORK . . . 6
2.1. Classification of Functional vs. Non-functional Requirements . . . 6
2.2. Automated Classifiers in Requirements Engineering . . . 7
2.3. Automated Feature Selection . . . 9
3. IMPLEMENTATION . . . 10
3.1. Preparing the Data sets . . . 10
3.2. Tagging . . . 13
3.3. Method . . . 16
3.3.1. Training Approach . . . 16
3.3.2. Metrics and tools . . . 16
3.4. Reference Classification based on Kurtanović and Maalej's Study . . . 18
3.5. Classification using Dependency Parsing and Linguistic Features . . . 26
3.5.1. Feature Creation with Dependency Parsing and Linguistic Features 26
3.5.2. Feature selection via interpretable ML . . . 30
3.5.3. Enhancing the Feature Selection with Features for Root Verbs . 34
3.5.4. Experimental Feature Selection with Automated Feature Selection 39
3.6. Threats to Validity . . . 55

4. CONCLUSIONS 56
REFERENCES 58
APPENDIX A: FEATURE SETS 64
APPENDIX B: FUNCTIONAL FEATURES PER FEATURE SET 65
APPENDIX C: QUALITY FEATURES PER FEATURE SET 71

LIST OF FIGURES

Figure 3.1. Example Data Set. 15

Figure 3.2. Illustration of the ROC plot and of the AUC. 19

Figure 3.3. ROC plots for the top 500 features: F (top-left), Q (top-right), OnlyF (bottom-left), OnlyQ (bottom-right). 24

Figure 3.4. ROC plots for the top 100 features: F (top-left), Q (top-right), OnlyF (bottom-left), OnlyQ (bottom-right). 25

Figure 3.5. Example Requirements from data sets showing the dependency types. 27

Figure 3.6. Interactive rule visualization interface of RuleMatrix. 34

Figure 3.7. ROC plot for F with the final 17 features. 38

Figure 3.8. ROC plot for OnlyQ with the final 17 features. 39

Figure 3.9. FFS - ROC Plot for F with selected features by FFS from FS 3. . 40

Figure 3.10. FFS - ROC Plot for Q with selected features by FFS from FS 3. . 41

Figure 3.11. Genetic Algorithm - ROC Plot for F with selected features by GA from Feature Set 3. 42

Figure 3.12. Genetic Algorithm - ROC Plot for Q w ith selected features by GA from Feature Set 3. 43

Figure 3.13. AutoVIML - Feature Importances for predicting F. 44

Figure 3.14. AutoVIML - Feature Importances for predicting Q. 44

Figure 3.15. AutoVIML - ROC Plot for F with selected features by AutoVIML from Feature Set 3. 44

Figure 3.16. AutoVIML - ROC Plot for Q with selected features by AutoVIML from Feature Set 3. 45

Figure 3.17. RFE - Feature Importances for predicting F. 46

Figure 3.18. RFE - Feature Importances for predicting Q. 46

Figure 3.19. Recursive Feature Selection - ROC Plot for F with selected features by RFE from Feature Set 3. 47

Figure 3.20. Recursive Feature Selection - ROC Plot for Q with selected features by RFE from Feature Set 3. 47

LIST OF TABLES

Table 3.1. Data sets 12

Table 3.2. Overview of the tagged data sets 15

Table 3.3. Project-fold splitting of the PROMISE NFR dataset. 17

Table 3.4. Descriptive statistics of Top 100 features for our reproduction of [1] for the classifier targets: F and Q. 20

Table 3.5. Top 100 features for our reproduction of [1] for OnlyF and OnlyQ. 21

Table 3.6. Top 500 features for our reproduction of [1] for F and Q. 22

Table 3.7. Top 500 features for our reproduction of [1] for OnlyF and OnlyQ. 23

Table 3.8. Precision, Recall, and F1-score with the three feature sets. 31

Table 3.9. Results with the features selected by Skoperules and Rulematrix. . 35

Table 3.10. The final feature set. 36

Table 3.11. Results with additional features for root verbs. 38

Table 3.12. AUC Comparison of all feature sets for F classifier target. 49

Table 3.13. AUC Comparison of all feature sets for Q classifier target. 52

LIST OF ACRONYMS/ABBREVIATIONS

AUC	Area under ROC Curve
CP	Clause and Phrase
DL	Deep Learning
FFS	Forward Feature Selection
FR	Functional Requirement
FS	Feature Set
ML	Machine Learning
NFR	Non-functional Requirement
POS	Part of Speech
RE	Requirements Engineering
RFE	Recursive Feature Elimination
ROC	Receiver Operating characteristic
SULOV	Searching for Uncorrelated List of Variables
SVM	Support Vector Machine

1. INTRODUCTION

Requirement classification is the categorization of requirements into different classes to form a logical group for their usage in the design, implementation, decision making, and other processes [2]. With classification, we can decompose a system into sub-components of related requirements, and we can define the relationship between these sub-components.

Requirements are generally classified as functional requirements and non-functional requirements. Glinz [3] defines a functional requirement as "a requirement concerning a result of behavior that shall be provided by a function of a system or of a component or service, where a non-functional requirement as a quality requirement or a constraint". A functional requirement specifies what function the system should perform and a non-functional requirement specifies how the system should perform that function.

Machine learning is widely used concept in requirements classification. Automated classifiers have been used for many purposes, including the classification of non-functional requirement [1, 4–6], the user feedback analysis to distinguish between bugs, features, and complaints [7], or the distinction of requirements from information in big requirements specification data sets [8].

1.1. Problem Statement

Classification is a fundamental challenge in requirements engineering. Glinz [9] identified three main challenges like the lack of agreement on *i.* the definition, *ii.* the sub-classification, and *iii.* the representation of non-functional requirements. The classification of requirements is a multi-level problem. First, it starts with the identification of whether a text is a requirement or information. If it is a requirement, it needs to be decided whether it is a functional requirement or a non-functional requirement.

If the requirement includes quality aspects, we need to find out what kind of a quality requirement that is, i.e. security, usability, etc.

In practice, there are even further problems in the classification of the requirements. The requirements are mostly tagged and classified by users, and the users mostly have different perceptions and do not reach a consensus on the classification. Users' experience, knowledge, as well as state of mind have an impact on this problem: a security expert approaches the requirements from a security perspective, meanwhile, a business expert may identify the same requirement as functional and a programmer as reliability. It is difficult to achieve a fully correct classification. In most cases, there is no single answer, neither. The classification process is not a binary classification: a requirement can include quality aspects from multiple areas, such as a requirement can be functional as well as non-functional at the same time. When being non-functional, it can be a security as well as a usability requirement. This makes the agreement as well as classification quite difficult.

Many requirements classification approaches exist, and they mainly differ from each other in the creation and selection of the features, the classifiers they employ (SVM, decision trees, Naïve Bayes, etc.), the metrics that they use to interpret the outputs, or the data sets that are used for training and validation [5]. The main commonality is that the most effective classifiers [1] rely on a high-dimensional word feature set: a large number of features are used to guide the classifier (100 to 500 in [1]), and these features are at the word level, e.g., text N-grams or Part-of-Speech (POS) n-grams.

An N-gram model is the prediction of the occurrence of a word compared to the occurrence of its N – 1 previous word, defining how far the prediction goes back in the past sequence of words to predict the next word. For example, a bigram model (N=2) predicts the occurrence of a word given only its previous word (as N – 1 = 1 in this case). Similarly, a trigram model (N = 3) predicts the occurrence of a word based on its previous two words (as N – 1 is 2 in this case).

A POS (part-of-speech) tag is a label assigned to each word in a text to label the words that have specific purposes such as nouns, verbs, and adjectives.

Highly dimensional feature sets often lead to classifiers that suffer from *overfitting* [10]: their performance is excellent on the training set, but it degrades significantly with other data sets. The high number of features in the feature sets reduces the interpretability of those classifiers. Since the algorithm's decision rule may have a combination of hundreds or thousands of features. it is very difficult to analyze why the classifier performs well or poorly. Furthermore, the features represent the requirements at the word level, rather than focusing on the meaning of a requirement.

1.2. Purpose of the Study

In this study, we target the classification of functional and non-functional requirements. Theoretical foundations on this topic [5] such as the distinction between functional requirements, non-functional requirements, quality constraints, etc., are widely discussed in RE [9, 11]. We propose a classification approach that uses smaller sets of linguistic features based on dependency types which are interpretable [12]. We use interpretable machine learning tools to select the features [13] which help to achieve a deeper understanding of the implicit classification rules of automated classifiers. We show that it is possible to build more general classifiers using high-level linguistic features, which apply to different data sets with a little performance degradation as well as easier to inspect. We also show that interpretable ML can contribute to the scientific discussion on classification problems through explainable classifiers.

Our study was published in the IEEE 27th International Requirements Engineering Conference (RE'2019) Proceedings [14]. As being one of the authors, this thesis is mainly based on our study and we extend our findings with additional data sets and with automated feature selection algorithms.

1.3. Approach

In this study, we take the concepts of Li *et al.* [11] as the baseline and build our distinctions in two aspects:

- *Functional aspect (F)*: a requirement includes either a functional goal or a functional constraint;
- *Quality aspect (Q)*: a requirement includes a quality goal or a quality constraint.

The decision on the functional aspect is independent of the decision on the quality aspect; thus, a requirement can possess only F aspects, only Q aspects, both aspects (F+Q), or none of them, which indicates that requirement is information [8].

Although Hey *et al.* [15] use the same data sets and achieved slightly better results, we do not use NorBERT, as it is important to be able to analyze and explain the results, especially when working on different domains and different terminology.

The overall approach can be summarized as follows:

- Building on recent studies about the nature of functional and quality requirements [11], we take the approach that every requirement can possess both functional and quality aspects. With the other authors of the paper, we tagged a set of more than 2500 requirements, which includes the PROMISE data set [16] and requirement data sets from thirteen industrial projects.
- To define the baseline, we reconstructed a word-level classification approach that relies on a high-dimensional feature set [1] and applied it to the revised classification problem. The results confirm its excellent performance on the training set and show limited generality when applied to the industrial data sets.
- We develop a classification approach based on smaller sets of linguistic features such as dependency types [12]. We selected the features by using interpretable machine learning tools [13] that provide if-then-else rules.

- We compare the new approach with the state-of-the-art [1] both quantitatively and qualitatively. The results show that the performance of the two approaches is generally comparable, while classifiers that select the features using interpretable ML can be more easily interpreted.
- We construct a set of automated feature selection algorithms and apply several combinations to our feature sets to find the optimal feature sets, and compare the results.

1.4. Organization

This thesis is structured as follows. Section II overviews the related work and ´previous studies on non-functional requirements and automated classifiers which define theoretical framework. Section III presents the methodology used, describes our reference classifier by assessing its performance, describes the creation and selection of the feature set that we use in our approach, and experiments with the automated feature selection algorithms. Section IV concludes the findings and discusses the future work.

2. BACKGROUND AND RELATED WORK

In this section, the previous related works about the classification of requirements have been presented.

2.1. Classification of Functional vs. Non-functional Requirements

Glinz [9] proposes a classification based on the taxonomy of the terms that are based on concerns, which are the matter of interests in the system. According to Glinz, the set of all requirements of a system is partitioned into functional requirements, performance requirements, specific quality requirements, and constraints. Glinz defines a functional requirement as a requirement that pertains to a functional concern. A performance requirement is a requirement that pertains to a performance concern. A specific quality requirement is a requirement that pertains to a quality concern other than the quality of meeting the functional requirements and a constraint is a requirement that constrains the solution space beyond what is necessary for the meeting. This approach does not deal with project and process requirements.

Li *et al.* [11] adopt a quality-oriented approach to model nonfunctional requirements as quality goals. Quality is a basic perceivable or measurable characteristic that inheres in and existentially depends on its subject, and it has a quality type (e.g., usability) and a quality value (e.g., acceptable). Quality goals map a quality type to a quality value. A quality constraint is similar but maps to measurable values. We take the approach of Li et al. as our baseline and propose a simplification that combines their concepts into two aspects:

- Functional aspect (F): a requirement includes either a functional goal or a functional constraint;
- Quality aspect (Q): a requirement includes a quality goal or a quality constraint.

The decision on the functional aspect is independent of the decision on the quality aspect; thus, a requirement can possess only F aspects, only Q aspects, both aspects (F+Q), or none. In the last case, the requirement denotes information [8].

Casamayor *et al.* [17] extract non-functional requirements from natural language text. Mahmoud [18] associates keywords with classes of non-functional requirements, calculates co-occurrence of these terms and creates clusters using this metric. Then, the clusters are classified under sub-categories of non-functional requirements with an average accuracy of 73%.

2.2. Automated Classifiers in Requirements Engineering

Zhang *et al.* [19] investigate different ML techniques to automatically classify non-functional requirements and conclude that individual words are the best index terms in text to indicate non-functional requirements.

Hussain *et al.* [20] use linguistic features such as cardinals, adverbs, modals, and so on to train a classifier that identifies non-functional requirements in software requirements specifications documents. The approach is also trained and tested on PROMISE.

Singh and Sharme [21] combine automated identification and classification of requirements into non-functional requirement sub-classes with the help of a rule-based classification technique using thematic roles. They identify the priority of the extracted non-functional requirements according to their occurrence in multiple classes. Application of this method within the PROMISE corpus results in F1-measure of 97%.

Abad *et al.* [6] introduce an approach for preprocessing requirements that standardizes and normalizes requirements before applying classification algorithms. They show that preprocessing and standardization of the PROMISE NFR dataset improves the results of the classification.

This result is not surprising since PROMISE includes requirements from 15 distinct projects written in different styles.

Hey *et al.* [15] address similar problems and use NoRBERT that fine-tunes BERT, a language model that has proven useful for transfer learning to overcome the same issues. They highlight the problem of poor generalization and try to improve the performance of classification when applied to unseen projects. They investigate the performance of NoRBERT using multi-class classification and show that training data evenly distributed over the classes is a major factor for better results and difficulties rise mostly on highly imbalanced data sets.

Contents of a requirements specification document can not be considered as requirements only. The document also includes information such as constraints and domain assumptions. Vogelsang and Winkler [8] introduce an approach to automatically classify the content elements of a natural language requirements specification document as "requirement" or "information" using convolutional neural networks with high precision.

Navarro-Almanza *et al.* [22] use Deep Learning (DL) to classify software requirements using CNNs that have been the state of the art in other natural language-related tasks. They also use the PROMISE corpus in their evaluation and achieved precision, recall, and f-measure values of 0.80, 0.785 and 0.77 respectively.

Kurtanović and Maleej [1] study how accurately and automatically requirements can be classified as functional (FR) and non-functional (NFR) with supervised machine learning using meta-data, lexical, and syntactical features. They also evaluate how accurately various types of NFRs can be identified, in particular usability, security, operational, and performance requirements.

2.3. Automated Feature Selection

Creating and selecting the right features is one of the difficult parts of automated classifiers. All features that exist in a data set do not have the same priority. Some features are redundant or some features are more important than others. Selecting the right features are also important to speed up the learning and to improve the quality of the classifier [23]. The classifiers rely mainly on domain knowledge and instincts of the developers, therefore the final feature set is limited by human subjectivity and also by time.

Feature selection is an optimization problem that selects the most relevant features from an original feature set to increase the performance of classification [24]. Feature selection algorithms have different evaluation criteria and the filter model, the wrapper model, and the hybrid model are the widely used models [25]. The filter model is based on the general characteristics of the data to evaluate and select feature subsets. The wrapper model has a single predefined algorithm and it selects the features better suited to the mining algorithm to have an improved mining performance, but it is computationally heavier than the filter model. The hybrid model tries to use the advantages of the both models by using their different evaluation criteria in different search stages.

Khurana *et al.* [26] performs automatic feature engineering by exploring different feature selection scenarios in a hierarchical and non-exhaustive manner, and tries to increase the accuracy through deep mining. Additionally, the system allows users to specify the domain or data-specific choices to prioritize the exploration. The system, Cognito, is capable of handling large data sets through sampling and built-in parallelism and integrates well with a state-of-the-art model selection strategy.

3. IMPLEMENTATION

We select a high-dimensional, word-level feature set [1] as a reference in terms of classification performance and interpretability. This work is described in detail so that we were able to reconstruct it and demonstrated very good results, therefore we believe it would be an excellent reference set. After reconstructing the feature set, we use this set in an automated classifier, train the classifier on a data set (PROMISE NFR [27]), and assess the performance on heterogeneous data sets. We also implement additional experimental feature selection algorithms to select features and we compare the results with our feature set using the same data sets, classifier, and training approach.

3.1. Preparing the Data sets

We first started the study with 8 data sets. The number of requirements per data sets differs from 625 to 62, although one of them is a collection of 15 distinct projects. Out of the eight data sets, two data sets are not open to the public. After the first results, we extended the data sets and included 6 more data sets. Table 3.1 summarizes the information on the datasets.

- *PROMISE* [27] is a collection of 625 requirements from 15 different projects, created and classified by students, and has previously been used to train and test other requirements classification approaches (e.g. [1, 11]).
- The *ESA Euclid* data set is a subset of system requirements of the European Space Agency for the Euclid mission [28]; and includes 236 requirements mainly elicited from sections: reliability and safety, safe mode, altitude and orbit control, propulsion, telemetry, tracking, and command.
- The *Dronology* dataset [29] has 97 system requirements for Unmanned Aerial Systems (UASs).
- The *ReqView* data set [30] details the requirements specification for the ReqView requirements management tool.

We have converted the format of the requirements from separate modal verbs and adverbial clauses to full sentences.

- *Leeds library* data set includes requirements for Leeds's University Library online management system and is documented in an online spreadsheet [31]. We have removed lines of text that are not related to requirements.
- *Helpdesk* data set is a private data set from IT domain, and contains 172 user requirements for implementing an off-the-shelf help desk system.
- User mgmt data set is another private data set, again from the IT domain and includes 138 requirements for a bespoke user account request and management application.
- *WASP* data set [32] consists of requirements for the Web Architectures for Services Platforms (WASP) application.
- The *rds4* data set is an extract of example non-functional requirements from the book "The Quest for Software Requirements" [33, 34].
- *OAppT* data set is a private data set from the IT domain and the data set contains 140 requirements for implementing an off-the-shelf Online Application Tool for recruitment.
- *RepReq* data set has 75 user requirements for an off-the-shelf Reporting Application. This data set is also private data set.
- The *rds8* data set has 291 requirements and includes the requirements of an internal project from a NATO agency regarding a streaming platform [35].
- The *rds9* data set has 228 requirements and includes the requirements for Electronics Record Management Systems, a project led by Public Record Office in the UK [36].
- The *rds12* data set has 148 requirements and includes the requirements for Michigan Department of Transportation's Vehicle Infrastructure Integration Data Use Analysis and Processing System [37].

Table 3.1. Data sets.

ID	Name	Rows	Availability
DS_1	PROMISE [16]	625	Public
DS_2	ESA Euclide	236	Public
DS_3	non-disclosed	172	Private
DS_4	non-disclosed	138	Private
DS_5	dronology	97	Public
DS_6	ReqView	87	Public
DS_7	Leeds Library	87	Public
DS_8	WASP	62	Public
DS_9	rds4	130	Public
DS_{10}	OAppT	140	Private
DS_{11}	RepReq	75	Private
DS_{12}	rds8	291	Public
DS_{13}	rds9	228	Public
DS_{14}	rds12	148	Public

3.2. Tagging

The taggers for the requirements are the authors of our paper. They are all experienced in the requirements engineering. Dr.Fatma Başak Aydemir is an assistant professor in the Department of Computer Engineering at Boğaziçi University, Turkey. She leads the requirements engineering group. Dr.Fabiano Dalpiaz is an assistant professor in the Department of Information and Computing Sciences at Utrecht University in the Netherlands. He is the principal investigator in the department's Requirements Engineering lab. Dr.Davide Dell'Anna is a postdoctoral researcher in the Department of Control and Operations of Delft University of Technology and he obtained a Ph.D. from Utrecht University recently, with a thesis in the fields of Artificial Intelligence and Requirements Engineering, on data-driven supervision of autonomous systems. Finally, I am working in the IT sector for the last 24 years, and have been actively involved in the procurement, implementation, and operation of many IT systems including gathering and eliciting the related functional and non-functional requirements of those systems.

For the tagging, we applied a supervision learning method. Before the tagging process, we extracted a random set of requirements from the data sets. We had a meeting to review and discuss the classification approach and agreed on the guidelines. Then we created 2 sub-groups, each composed of 2 taggers. The groups tagged each set independently. For each requirement in the set, we classified it as a requirement or not, and if it is a requirement, we classified it as functional or quality, or both. where the meanings are whether the requirement includes functionality and quality, or both respectively. Then, the taggers organized reconciliation meetings to go over the differences in tagging and re-conciliated the results. If the taggers failed to convince each other, a third tagger was consulted for the final tagging. The taggers went overall differences and managed to resolve them. For example, in PROMISE data set, an agreement was not reached for the requirement "The system shall refresh the display every 60 seconds.' where one tagger tagged the requirement as functional, and the other tagged as both functional and non-functional.

After the discussion with the other team, it has been reconciled as functional.

There were some tricky cases, in which one 'requirement' clearly refined another. For example, "The application shall display a list of open vacancies." is defined by "The list of open vacancies shall display Vacancy number", "The list of open vacancies shall display Vacancy title", etc. Therefore, there were discussions about whether the latter two are requirements, at the end, we tagged them as functional. There were quite discussions about the usability requirements that define user interface design choices, for example, the requirements "The following fields shall be a Yes/No tick box field: "Have you applied for a job here before?"." Although there is a clear reference to the user interface, the sub-characteristics of usability are not reflected. Therefore discussions took place on whether this would just be a functional requirement but also a quality requirement with consideration for usability in a generic sense. User protection error was another tricky case, especially for input validation to avoid errors. Generally, whenever there is user protection from errors, there is a function to implement: the one that validates the input, which makes them both functional and quality. In some requirements, the use of the word "only" caused doubts and confusion. For example, "The report shall only display information for Staff Positions" where the requirement has a function to display the staff positions, however, the word "only" brings a quality constraint. The requirements which include the verb "Manage" are considered as not functional, because they are too abstract in saying what has to be implemented unless they are further specified. Figure 3.1 illustrates an example view of the tagged data set.

Table 3.2 summarizes the output of the tagging process. The data sets are ordered by the number of requirement rows. As described, the taggers assigned F and Q tags. Using these two tags, we then automatically calculated whether the row is tagged with only F (OnlyF), only Q (OnlyQ), both F and Q (F+Q), or neither of them (¬R). The reconciled classification is then used for training and testing the classifiers.

ProjectID	RequirementText	IsFunctional	IsQuality
1	User shall be able to use the application on Windows, Linux and OSX.	0	1
1	User shall be able to use the application without installation of any additional SW except the web browser	0	1
1	ReqView shall use REST API to download / upload a requirements document from / to Google Drive web service	1	1
1	The data exchange via Internet shall be done via encrypted https protocol	0	1
1	User shall be able to create a new empty document	1	0
1	User shall be able to save the current document containing unsaved changes when creating a new document	1	0
1	User shall be able to open a document file stored on a local user's drive	1	0
1	User shall be able to save the current document containing unsaved changes when opening another document	1	0
1	User shall be able to save modified document as a file stored on user's local drive	1	0
1	User shall be able to save the current ReqView document as a template file stored on user's local drive	1	0
1	The template file shall contain information about document structure, definition of custom attributes and traceability definition	1	0
1	User shall be able to create a new document using a predefined or custom template	1	0
1	User shall be able to encrypt saved file by a strong industry standard encryption and with a user defined password	1	1
1	User shall be able to authorize ReqView with his Google Drive	1	1
1	User shall be able to open a document stored on user's Google Drive by providing its name	1	1
1	User shall be able to save the current document to user's Google Drive providing its name and password for encryption	1	1
1	User shall be able to upload a new version of a document opened from user's Google drive	1	1
1	User shall be able to import a structured requirements document from MS Word	1	1
1	User shall be able to import a table from MS Excel	1	1
1	User shall be able to export document to PDF with the same layout as displayed in the Requirements Table	1	1
1	User shall be able to export ReqView document to HTML using a custom template	1	1
1	User shall be able to export ReqView document to MS Word	1	1
1	User shall be able to show and hide Table of Contents (TOC) containing overall document structure visualized as a tree	1	1
1	User shall be able to navigate in the document by clicking at TOC sections	1	0

Figure 3.1. Example Data Set.

Table 3.2. Overview of the tagged data sets.

Dataset	**Public**	**Rows**	**F**	**Q**	**Only F**	**Only Q**	**F+Q**	**R**
PROMISE	Yes	625	310	382	230	302	80	13
rds8	Yes	291	135	233	45	143	90	13
ESA Euclid	Yes	236	91	211	23	143	68	2
rds9	Yes	228	163	149	65	51	98	14
RepReq	No	75	40	47	20	27	20	8
Helpdesk	No	172	143	51	121	29	22	0
rds12	Yes	148	138	110	37	9	101	1
User mgmt	No	138	126	25	113	12	13	0
OAppT	No	140	84	56	55	27	29	29
rds4	Yes	130	15	117	3	105	12	10
Dronology	Yes	97	94	28	68	2	26	1
ReqView	Yes	87	75	32	54	11	21	1
Leeds library	Yes	85	44	61	23	40	21	1
WASP	Yes	62	55	19	42	6	13	1
Totals		2514	1513	1521	899	907	614	94

3.3. Method

In this section, we explain the approaches we took on the training of the data set as well as describe the metrics and the tools we use on the evaluation of the results.

3.3.1. Training Approach

Our training approach considers 5 different conditions: The first one is the full usage of PROMISE (*fit*)bas the training data set. It tries to answer the question, how well does the high-dimensional classifier, which is trained on a high-dimensional feature set, fit the training data set that is used to construct the classification model. The second one is the 75%-25% splitting of PROMISE (*75/25*) which is applied by randomly splitting the data set into two: 75% of the entries are used to train the classifier, the remaining 25% for testing it. The third one is the k-fold cross-validation of PROMISE (*kfold*) where The data set is split into k evenly sized parts (*folds*), and the classifier is tested k times by training it on the *k-1* folds and testing it on the selected k^{th} fold. We use a stratified k-fold, which ensures a similar class ratio (positive/negative) in each of the folds. The fourth one is the project-level cross-validation (*pfold*) which choosing some projects within the data set as a training set, and the others as the test set. Since PROMISE consists of 15 projects, we used 12 of them as the training set, and 3 as the test set. To increase generality, we produce 10 variants of such partitioning such that every partition has at least 100 requirements and has a balanced F and Q ratio. Moreover, we ensure that two projects co-occur in at most one test set. At last, we use the industrial data sets (*ind*) where the classifier trained on PROMISE is evaluated on the industrial data sets of Section 3.1.

3.3.2. Metrics and tools

The study of Kurtanović and Maalej [1] had a single binary classification problem (F vs. Q) and used metrics for each of the two classes F and Q.

Table 3.3. Project-fold splitting of the PROMISE NFR dataset.

Proj	**Size**	**F**	**Q**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	28	19	15		x		x						
2	40	22	26						x			x	
3	80	45	33	x				x					
4	55	26	37									x	x
5	73	37	37		x				x				
6	74	31	49			x				x			
7	23	15	10								x		x
8	93	38	74				x				x		
9	24	17	7	x						x			
10	53	43	15			x							x
11	13	3	13	x					x				
12	22	8	22		x			x					
13	19	3	18			x					x		
14	16	3	14				x			x			
15	12	0	12					x				x	
			Size	117	123	146	137	114	126	114	135	107	131
Totals			F	65	64	77	60	53	62	51	56	48	84
			Q	53	74	82	103	67	76	70	102	75	62

We extend this approach and our framework of Section 3.1 leads to four binary classification problems to be studied:

- **F:** does a requirement possess functional aspects?
- **Q:** does requirement possess quality aspects?
- **OnlyF:** does a requirement possess functional aspects but no quality aspects?
- **OnlyQ:** does a requirement possess quality aspects but no functional aspects?

For each classification problem, we use the most popular metrics in the RE literature to assess the performance of binary classifiers. These are the precision, recall, and F1 score. Additionally, we use the receiver operating characteristic (ROC) plot and its associated metric, the area under the ROC curve (AUC).

ROC plots [38] are 2-dimensional charts that show the trade-off between recall (y-axis) and false negative rate (x-axis). In ROC plots, classifiers are represented as a line that is plotted by calculating recall and the false positive rate at different levels of the discrimination threshold. The discrimination threshold is the value in the $[0, 1]$ range that a classifier uses to determine when a data item should be classified as a positive. While this threshold is set to 0.5 by default for a binary classification problem, it can be adjusted to alter the sensitivity to false positives. Better classifiers are characterized by a curve that stays closer to the top-left corner. The ROC plot provides a single performance metric for a classifier, the AUC [39], that measures the degree of separability between the two classes. A perfect classifier has an AUC of 1.0, an always-wrong classifier has an AUC of 0.0, and a classifier with random performance has an AUC of 0.5. Figure 3.2 [14] illustrates these notions.

3.4. Reference Classification based on Kurtanović and Maalej's Study

The original classifier by Kurtanović and Maalej [1] is not available online and we could not get access to a working copy, however, as being one of the reasons to select it as a reference, the original publication [1] is relatively clear on the feature set.

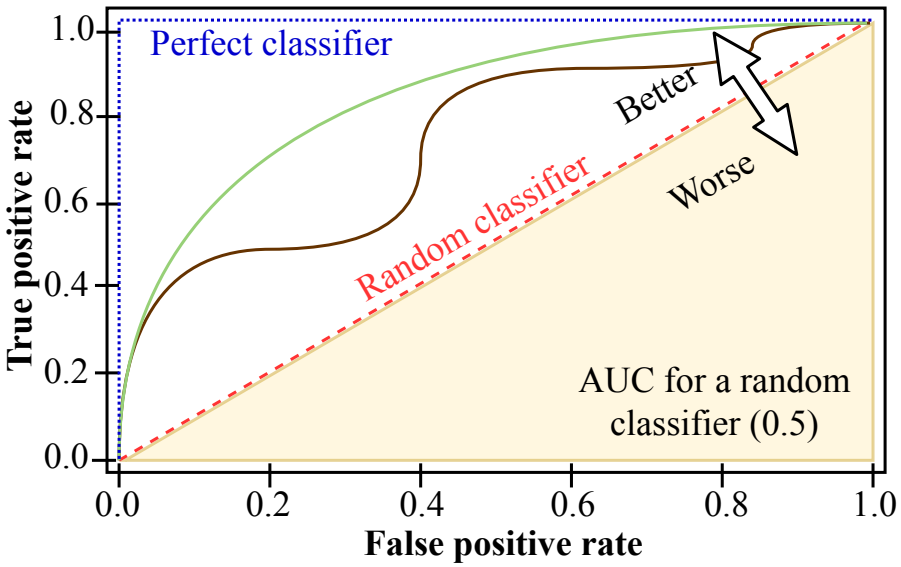


Figure 3.2. Illustration of the ROC plot and of the AUC.

We complemented this knowledge with the classifier that the same authors developed to classify app reviews [40], whose code is partially available online [41].

While reconstructing the original, we applied a few minor modifications to the original version:

- We built the parse trees using Berkley's *benepar* library, a state-of-the-art constituency parser that outperforms [42] the Stanford parser used in [1].
- We could not reproduce the feature "CP features", which was described as "unigrams of part of speech (POS) tags on the clause and phrase-level (CP)". This text was insufficient for us to make a correct re-implementation.
- We did not use the data set taken from Amazon software reviews that the authors had used to artificially balance the minority class of NFRs.
- We tested the classifier in its configuration with the 500 most informative features (best performance) and the 100 most informative features.
- Our four classification problems entailed that we had to train the classifier four times, for the most informative features depend on the target class: F, Q, OnlyF, OnlyQ.

The reconstructed code can be accessed through two publicly available Jupyter Notebooks [43].

Table 3.4. Top 100 features for our reproduction of [1] for F and Q.

Data set	**F**				**Q**			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
PROMISE train	0.877	0.897	0.886	0.95	0.919	0.955	0.922	0.97
PROMISE test	0.819	0.797	0.822	0.891	0.909	0.891	0.873	0.917
PROMISE k-fold	0.755	0.684	0.712	0.80	0.785	0.867	0.822	0.84
PROMISE p-fold	0.749	0.602	0.663	0.78	0.714	0.877	0.781	0.80
Industry (macro-avg)	0.710	0.658	0.654	0.668	0.599	0.625	0.577	0.602
Industry (std-dev)	±.23	±.15	±.08	±.06	±.24	±.14	±.13	±.08
ESA Euclid	0.477	0.451	0.597	0.59	0.898	0.706	0.665	0.54
Helpdesk	0.903	0.972	0.890	0.78	0.542	0.510	0.727	0.67
User mgmt	0.583	0.643	0.594	0.65	0.151	0.560	0.348	0.47
Dronology	1.000	0.670	0.680	0.77	0.370	0.607	0.588	0.66
ReqView	0.898	0.707	0.678	0.66	0.409	0.562	0.540	0.58
Leeds library	0.654	0.773	0.671	0.72	0.700	0.689	0.565	0.52
WASP	0.898	0.800	0.742	0.66	0.311	0.737	0.419	0.61
RepReq	0.579	0.825	0.587	0.688	0.625	0.319	0.453	0.554
rds4	0.190	0.533	0.685	0.681	0.925	0.838	0.792	0.68
rds8	0.600	0.422	0.601	0.619	0.837	0.794	0.711	0.637
rds9	0.805	0.607	0.614	0.625	0.622	0.497	0.474	0.501
rds12	0.988	0.587	0.608	0.689	0.910	0.555	0.628	0.717
OAppT	0.671	0.560	0.571	0.582	0.488	0.750	0.586	0.684

We analyzed the results from our experiments: precision, recall, and F1 score are reported in Table 3.4, Table 3.5, Table 3.6 and Table 3.7 with two high-dimensional feature sets that included the most informative features automatically selected through scikit-learn libraries: the top-100 and the top-500.

Table 3.5. Top 100 features for our reproduction of [1] for OnlyF and OnlyQ.

Data set	**OnlyF**				**OnlyQ**			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
PROMISE train	0.927	0.887	0.933	0.98	0.884	0.884	0.888	0.95
PROMISE test	0.870	0.870	0.911	0.94	0.896	0.852	0.873	0.91
PROMISE k-fold	0.766	0.630	0.681	0.86	0.741	0.798	0.766	0.82
PROMISE p-fold	0.752	0.475	0.573	0.81	0.683	0.794	0.728	0.81
Industry (macro-avg)	0.475	0.440	0.626	0.640	0.403	0.494	0.651	0.639
Industry (std-dev)	±.27	±.19	±.11	±.10	±.24	±.13	±.09	±.09
ESA Euclid	0.068	0.174	0.686	0.48	0.708	0.524	0.581	0.64
Helpdesk	0.785	0.843	0.727	0.69	0.368	0.241	0.802	0.72
User mgmt	0.872	0.301	0.391	0.63	0.622	0.578	0.610	0.61
Dronology	0.783	0.529	0.567	0.66	0.050	0.500	0.794	0.74
ReqView	0.639	0.426	0.494	0.60	0.333	0.636	0.793	0.70
Leeds library	0.250	0.217	0.612	0.51	0.645	0.500	0.635	0.73
WASP	0.850	0.405	0.548	0.64	0.176	0.500	0.726	0.66
RepReq	0.400	0.600	0.653	0.736	0.300	0.222	0.533	0.447
rds4	0.111	0.667	0.869	0.879	0.847	0.581	0.577	0.620
rds8	0.234	0.244	0.759	0.579	0.604	0.587	0.608	0.639
rds9	0.293	0.415	0.548	0.545	0.261	0.353	0.632	0.619
rds12	0.397	0.622	0.669	0.716	0.098	0.667	0.608	0.683
OAppT	0.500	0.273	0.607	0.661	0.226	0.519	0.564	0.474

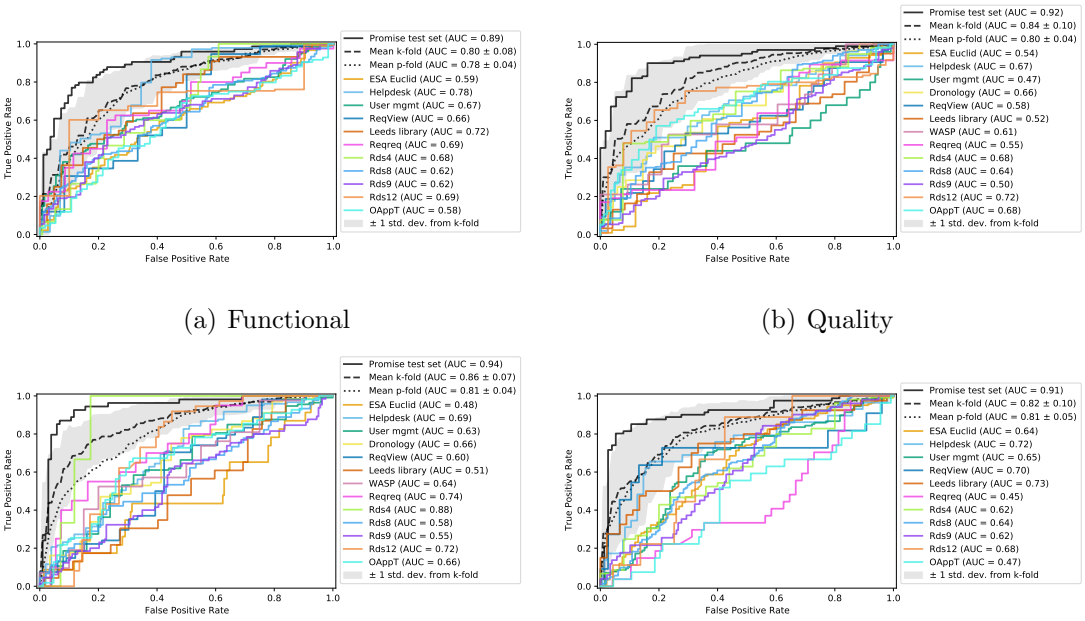
Table 3.6. Top 500 features for our reproduction of [1] for F and Q.

Data set	**F**				**Q**			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
PROMISE train	0.981	0.984	0.982	0.997	0.985	1.000	0.990	1.000
PROMISE test	0.795	0.784	0.803	0.90	0.910	0.901	0.879	0.95
PROMISE k-fold	0.819	0.742	0.774	0.87	0.817	0.909	0.858	0.89
PROMISE p-fold	0.805	0.699	0.742	0.85	0.752	0.917	0.823	0.85
Industry (macro-avg)	0.755	0.680	0.690	0.746	0.614	0.573	0.590	0.606
Industry (std-dev)	±.21	±.13	±.10	±.11	±.22	±.13	±.12	±.08
ESA Euclid	0.526	0.549	0.636	0.660	0.904	0.711	0.674	0.593
Helpdesk	0.914	0.972	0.901	0.865	0.767	0.451	0.797	0.736
User mgmt	0.898	0.627	0.780	0.854	0.191	0.680	0.420	0.575
Dronology	1.000	0.734	0.742	0.968	0.452	0.500	0.680	0.710
ReqView	0.961	0.653	0.678	0.858	0.410	0.500	0.552	0.611
Leeds library	0.698	0.682	0.682	0.729	0.667	0.557	0.482	0.449
WASP	0.915	0.782	0.742	0.639	0.344	0.579	0.532	0.603
RepReq	0.537	0.725	0.520	0.609	0.615	0.340	0.453	0.587
rds4	0.265	0.600	0.762	0.752	0.932	0.821	0.785	0.700
rds8	0.613	0.481	0.619	0.676	0.852	0.768	0.708	0.634
rds9	0.788	0.822	0.715	0.734	0.732	0.477	0.544	0.588
rds12	0.973	0.514	0.534	0.711	0.667	0.600	0.480	0.523
OAppT	0.728	0.702	0.664	0.642	0.456	0.464	0.564	0.566

Table 3.7. Top 500 features for our reproduction of [1] for OnlyF and OnlyQ.

Data set	**OnlyF**				**OnlyQ**			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
PROMISE train	1.000	0.987	0.995	1.00	0.990	0.964	0.978	1.00
PROMISE test	0.863	0.815	0.892	0.96	0.859	0.753	0.809	0.90
PROMISE k-fold	0.818	0.674	0.732	0.90	0.785	0.808	0.795	0.82
PROMISE p-fold	0.816	0.515	0.616	0.86	0.745	0.802	0.770	0.81
Industry (macro-avg)	0.476	0.636	0.645	0.658	0.466	0.425	0.716	0.688
Industry (std-dev)	±.29	±.21	±.10	±.10	±.30	±.25	±.12	±.14
ESA Euclid	0.149	0.478	0.682	0.634	0.693	0.490	0.559	0.633
Helpdesk	0.789	0.926	0.773	0.630	0.818	0.310	0.872	0.881
User mgmt	0.910	0.540	0.580	0.659	0.784	0.766	0.776	0.833
Dronology	0.820	0.603	0.629	0.719	0.000	0.000	0.866	0.816
ReqView	0.707	0.759	0.655	0.663	0.375	0.818	0.805	0.800
Leeds library	0.100	0.130	0.447	0.409	0.774	0.600	0.729	0.748
WASP	0.854	0.833	0.790	0.715	0.250	0.333	0.839	0.735
RepReq	0.378	0.700	0.613	0.710	0.158	0.111	0.467	0.362
rds4	0.074	0.667	0.800	0.892	0.886	0.590	0.608	0.654
rds8	0.230	0.378	0.708	0.652	0.653	0.566	0.639	0.712
rds9	0.351	0.815	0.518	0.585	0.378	0.275	0.737	0.660
rds12	0.354	0.622	0.622	0.611	0.106	0.556	0.689	0.560
OAppT	0.469	0.818	0.564	0.668	0.176	0.111	0.729	0.552

As a classifier, we used scikit-learn's Support Vector Machines (SVM) implementation that we executed with the linear kernel. This is the same classifier that was used in [1]. The ROC plot for top-500 is shown in Figure 3.3. and for top-100 is shown in Figure 3.4.



(a) Functional

(b) Quality

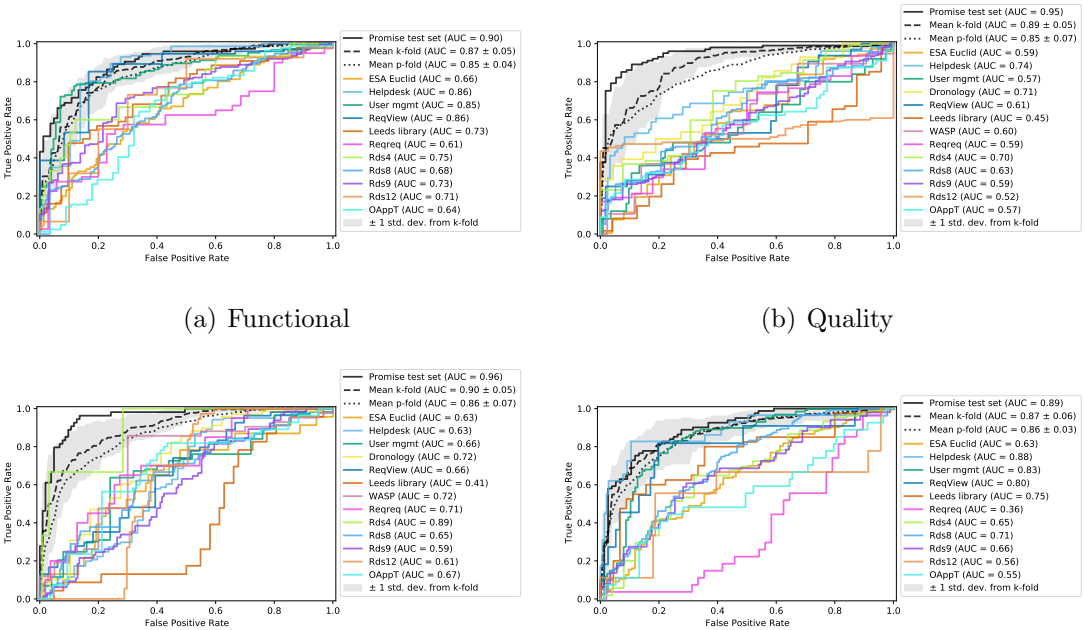
(c) Only Functional

(d) Only Quality

Figure 3.3. ROC plots for the top 500 features: F (top-left), Q (top-right), OnlyF (bottom-left), OnlyQ (bottom-right).

For the top-500 features setting in Table 3.6 and Table 3.7, the results for Q outperform those for F and OnlyF on the PROMISE-derived validation data sets (test, k-fold, and p-fold), especially in terms of recall. Conversely, in the industrial data sets, the results for F are way worse. However, In both top-100 and top-500, the classifiers perform best on the F class, especially in industry data sets. A possible reason is that this is the majority class in the dataset: 1513 out of 2514 rows possess a functional aspect, according to the taggers. An exception is the ESA Euclid dataset, which performs much better with Q.

The performance of the classifiers in identifying quality aspects (Q, OnlyQ) degrades considerably with the industry data sets.



(a) Functional

(b) Quality

(c) Only Functional

(d) Only Quality

Figure 3.4. ROC plots for the top 100 features: F (top-left), Q (top-right), OnlyF (bottom-left), OnlyQ (bottom-right).

Some key examples are Q, losing ∼0.3 in recall and ∼0.2 precision in both cases, compared to PROMISE, and OnlyQ, whose recall worsens ∼0.3 for top-100 and ∼0.2 for top-500. The performance for OnlyF and OnlyQ on industrial data sets shows high variance; the standard deviation for their precision is in the $[0.25, 0.31]$ range. An indication that the feature set did not lead to a general classifier.

Results for Q and OnlyF clearly illustrate that a significant degradation occurs when applying the classifier, trained on PROMISE, to more heterogeneous datasets like the industrial ones. The top-100 ROC plots show a similar situation, with the difference that two data sets (ReqView and User mgmt) outperform PROMISE-test in the F and OnlyQ settings. This is probably explained by class imbalance (75:11 and 126:12).

3.5. Classification using Dependency Parsing and Linguistic Features

With the mixed results of the industrial data sets after applying on our reference classifier, we use linguistic features like dependency types. Dependency types can improve the determination of whether a requirement contains functional or quality aspects. We test three selected feature sets on PROMISE and the industrial data sets through an SVM classifier configured as in Section 3.4 After introducing dependency parsing (Section 3.5.1), we use interpretable ML tools to select a feature subset that performs well and has a low size (Section 3.5.2).

3.5.1. Feature Creation with Dependency Parsing and Linguistic Features

Dependency parsing [12] is the task of identifying the grammatical structure of a sentence by determining the linguistic dependencies between the words. For instance, in the requirement 'The system shall clear the history every 2 hours', 'history' is the direct object (*dobj*) of the main (root) verb 'clear', 'shall' is the auxiliary verb (*aux*) that affects 'clear', '2' is the numeric modifier (*nummod*) of 'hours', etc. Each of these relationships is a *dependency type*.

Figure 3.5 shows some examples from from our data sets showing the dependency types included as features in our approach.

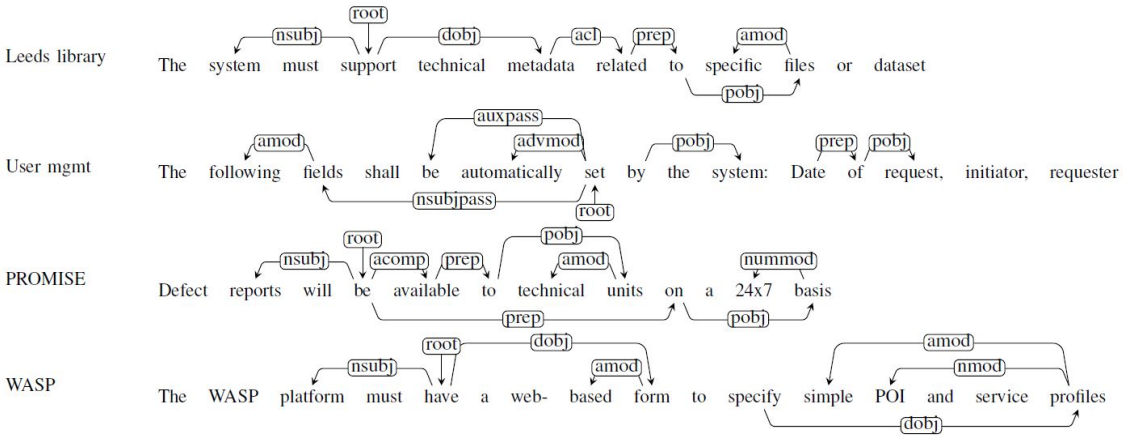


Figure 3.5. Example Requirements from data sets showing the dependency types.

For the creation of the features that we test, we adopt the following approach: We consider two categories of linguistic information: dependency types (e.g., *nummod* and *dobj*) and branches of the dependency tree of a requirement (e.g., a branch from the ROOT to a det passing through a dobj). For example, in the sentence 'clear the history', the branch type $ROOT \rightarrow dobj \rightarrow det$ connects the root verb 'clear' to the determinant 'the' via the direct object 'history'. We also consider the combination of two and three of them together (i.e., two/three dependency types or two/three branches together in the same requirement). In summary, we analyze six groups of linguistic features:

(i) single dependency types;
(ii) combinations of two dependency types;
(iii) combinations of three dependency types;
(iv) types of branches
(v) combinations of two types of branches together
(vi) combinations of three types of branches together

After determining the type of features to be considered, we analyze the set of requirements to determine the values (the existence) of these features and the results against the tags of the requirements (F, Q, F+Q, OnlyFor OnlyQ). We calculate the coverage (percentage) of each feature for these four tags. For example, the direct object dependency (*dobj*) dependency is present in 90% of all requirements, the direct object and determinant (*det*) occur together in 60% of OnlyFrequirements and 30% of OnlyQrequirements. We calculate the coverage difference between OnlyQ and OnlyF and sort the above-described features by the difference for each of the six groups: $\Delta = |cov(OnlyQ) - cov(OnlyF)|$, determining the ones that are more common in quality requirements but not in functional (highest Δ) and *vice versa*.

We inspect the 10 most significant (i.e. highest Δ) features for each category and determine several sets of features by combining differently some of them. The three of them performed better on average:

- *Feature Set 1:* The top 10 single dependency features plus other single dependency features that appeared in the top 10 combinations of two or three dependency types and that were not in the first ten selected. This resulted in 17 different dependency types.
- *Feature Set 2:* We take the top 10 features of each of the six groups, and we filter only those features with $\Delta > 0.2$. This resulted in 12 different linguistic features. To those we add additional features that were used in the literature [1, 20]:
  - (i) length of the requirement in characters;
  - (ii) the number of modals,
  - (iii) the number of adjectives,
  - (iv) the number of nouns,
  - (v) the number of adverbs,
  - (vi) the number of cardinals,
  - (vii) the number of comparative and superlative adjectives,
  - (viii) the number of comparative and superlative adverbs,
  - (ix) the number of words;

(x) the number of constituency parse subtrees in the requirement,

(xi) the height of the consistency parse tree,

(xii) the max height of the dependency tree of a sentence of a given requirement,

(xiii) the number of adverbial modifiers that link a verb to an adverb:

$$x \xrightarrow{advmod} y.\ x : VERB,\ y : ADV$$

(xiv) the number of adjectival modifiers that link a noun to an adjective, and the noun is not the subject of the sentence:

$$x \xrightarrow{amod} y.\ x : NOUN,\ y : ADJ,\ \nexists z.\ z \xrightarrow{nsubj} x$$

(xv) the number of adjectival phrases that complement a verb, and the adjectival phrase head differs from the word 'able':

$$x \xrightarrow{acomp} y.\ x : VERB,\ y : ADJ,\ y \neq 'able'$$

In total, Feature Set 2 includes 27 features.

- *Feature Set 3:* The 10 most significant features for each of the six categories of linguistic information above-described plus the features in Feature Set 2. In total this resulted in 60+12+3=75 different linguistic features.

Table 3.8 reports the results that we obtained with an SVC classifier using feature sets 1, 2, and 3, for *75/25* on PROMISE and for the average *ind*. We can notice how the results are comparable to the results obtained with the approach of Kurtanovic and Maalej, reported in Table 3.6 and Table 3.7. Differently from Kurtanovic and Maalej we only use a limited number of features and do not rely on lexical information but only on more general linguistic features. The advantage of using linguistic features comes from their interpretability and generality. Having a set of lexical information as features makes it hard to understand the reasons why a requirement is classified with a certain class.

Linguistic features, instead, give us insights into the rationale behind the classification. For instance, the presence of the three dependency types aux, dobj and det indicates the presence in a requirement (e.g., in "The system shall have the...") of a structure such as *shall have the*, which is typical of functional requirements. The same three dependencies capture also analogous requirements that however use different lexicon (e.g., *must* instead of *shall*), without the need of specifying every possible word as a feature.

3.5.2. Feature selection via interpretable ML

Best performing classifiers are the ones that are most complex and most difficult to explain their predictions. Although various visualizations have been developed, domain experts with little knowledge of machine learning have been quite neglected. With the growing adoption of ML techniques, there is an increasing trend towards making ML systems more transparent and interpretable [44]. Therefore, we employ two tools that facilitate the interpretation of ML classifiers.

Ming *et al.* [13] provide an interactive visualization technique to help users with little expertise in ML to understand, explore and validate predictive models. In this technique, they extract a standardized rule-based knowledge representation from its input-output behavior and design a RuleMatrix, a matrix-based visualization of rules to help users navigate and verify the rules and the black-box model.

SkopeRules [45] is another interpretable model that generates a list of rules but does not visualize them. Under the hood, SkopeRules applies a bagging estimator training where multiple decision tree classifiers are trained. The best rules are selected based on their performance trying to avoid duplicate rules. However, this approach mainly differs in the way that decision rules are chosen. Given the interpretable nature of the features that we identified, we aim at understanding better why requirements were classified as F or Q by our classifier.To do so, we use SkopeRules and RuleMatrix, and we identify rules of the form if-then-else that are used to classify the requirements.

Table 3.8. Precision, Recall, and F1-score with the three feature sets.

Set	**Data set**	**F**				**Q**			
		Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
FS 1	PROMISE test	0.67	0.73	0.70	0.73	0.84	0.75	0.75	0.78
	Ind.(macro-avg)	0.71	0.81	0.71	0.63	0.63	0.52	0.60	0.60
	Ind.(std-dev)	±.22	±.10	±.13	±.15	±.24	±.16	±.10	±.07
FS 2	PROMISE test	0.67	0.81	0.72	0.76	0.85	0.78	0.77	0.81
	Ind.(macro-avg)	0.78	0.93	0.80	0.74	0.58	0.28	0.62	0.60
	Ind.(std-dev)	±.17	±.08	±.13	±.09	±.24	±.15	±.13	±.08
FS 3	PROMISE test	0.70	0.74	0.73	0.76	0.86	0.83	0.80	0.84
	Ind.(macro-avg)	0.71	0.85	0.72	0.69	0.69	0.39	0.59	0.58
	Ind.(std-dev)	±.23	±.11	±.14	±.12	±.22	±.17	±.12	±.09
Set	**Data set**	**OnlyF**				**OnlyQ**			
		Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
FS 1	PROMISE test	0.61	0.74	0.75	0.78	0.73	0.68	0.70	0.73
	Ind.(macro-avg)	0.71	0.60	0.62	0.61	0.71	0.31	0.56	0.58
	Ind.(std-dev)	±.23	±.16	±.12	±.12	±.20	±.14	±.12	±.06
FS 2	PROMISE test	0.65	0.78	0.78	0.83	0.78	0.62	0.71	0.76
	Ind.(macro-avg)	0.83	0.82	0.77	0.77	0.76	0.17	0.59	0.57
	Ind.(std-dev)	±.18	±.12	±.10	±.07	±.20	±.10	±.16	±.11
FS 3	PROMISE test	0.67	0.65	0.77	0.85	0.75	0.69	0.72	0.76
	Ind.(macro-avg)	0.74	0.71	0.70	0.65	0.78	0.19	0.53	0.59
	Ind.(std-dev)	±.23	±.15	±.11	±.10	±.18	±.11	±.13	±.07

We report here an example of the set of rules identified using Feature Set 1 to classify functional requirements from the PROMISE data set. We report the simplest example of rules, even though not the best in performance, to avoid a too verbose section. SkopeRules identifies the following 3 exclusive rules. Requirements that fall in one of these rules are classified as functional, the remaining ones as not functional.

(i) $\neg advmod \wedge dobj \wedge nsubj \wedge \neg nsubjpass \wedge \neg nummod$

(ii) $\neg acl \wedge dobj \wedge \neg nmod \wedge nsubj \wedge \neg nummod$

(iii) $acl \wedge dobj \wedge nsubj \wedge \neg nummod \wedge pobj$

For instance, the first rule states that requirements, where there is a direct object dependency and the nominal subject of the sentence is not passive, and where there is neither a numeric nor an adverbial modifier, should be classified as functional. An example of such type of requirement is "The system shall display Events or Activities". Noting that with 3 simple rules that make use of only 8 different features, SkopeRules reaches a precision of 0.69 and a recall of 0.73 on the PROMISE test set, which is not that far from Kurtanovic and Maalej's results (see Table **??** and Table 3.7).

For the same feature set, RuleMatrix identifies the following rule. Each 'if' statement denotes one rule. Rules are therefore applied in cascade: a rule can fire only if none of the previous ones are fired. Additionally, a pair `[p,q]` in the rule indicates probability `q` for the class functional, and probability `p` for the class not functional.

We refer to the original paper [13] for a detailed explanation of the visualization. Each row corresponds to one of the above reported if statements. Such statements are applied in cascade, this is represented via the waterfall diagram on the left of the figure and the horizontal flow represents the amount of data satisfying the condition. Each column (e.g., `nummod` and `dobj`) represents one feature. Each feature has a grey area in the rows where such feature is considered and shows which values of the features are captured by the condition (e.g., condition 2 is activated for value 1 of feature *nummod*).

Column `Output` contains the probability of classifying a requirement with the class indicated by the color of the numerical value (e.g., requirements that satisfy condition 2 are classified as NFR with probability 1). Column `Fidelity` indicates the fidelity of such a rule with the outcome from the original classifier when the same conditions apply. Finally, column `Evidence` describes the amount of data correctly and wrongly predicted.

The three rules identified by RuleMatrix, for example, are similar to, but simpler than those produced by SkopeRule. RuleMatrix's ruleset uses only two features: *dobj* and *nummod*, and the rules reach a performance that is very similar to Feature Set 1's classifier on PROMISE for the F problem: a precision of 69% and a recall of 73% with only two features. Note that this is only 8% and 4% less than the results of the high-dimensional, word-level classifier with the top-500 features.

RuleMatrix and SkopeRules are powerful machine learning tools that allow the analysis of the output of a classifier. A simple visual exploration of the rules generated by such tools allows us to select a set of 15 most meaningful features starting from the three feature sets. Such set can be used for all the classification tasks we performed (F, Q, OnlyF and OnlyQ). Table 3.9 reports the results with such a feature set. The results show that the selected 15 most meaningful features, as expected, perform analogously to the three feature sets Feature Set 1, Feature Set 2, Feature Set 3. If compared to the results of Table 3.6 and Table 3.7, despite the performances are generally worst in the case of PROMISE, the performances in classifying functional (and especially only functional) requirements are better on average on the industrial data sets.

IF *dobj* THEN prob: [0.0133, 0.9867]
ELSE IF *nummod* THEN prob: [0.9981, 0.0019]
ELSE IF ¬*dobj* THEN prob: [0.0011, 0.9989]
ELSE DEFAULT prob: [0.9504, 0.0496]

IF *nummod* THEN prob: [0.9761, 0.0239]
ELSE IF ¬*dobj* THEN prob: [0.9503, 0.0497]
ELSE DEFAULT prob: [0.0011, 0.9989]

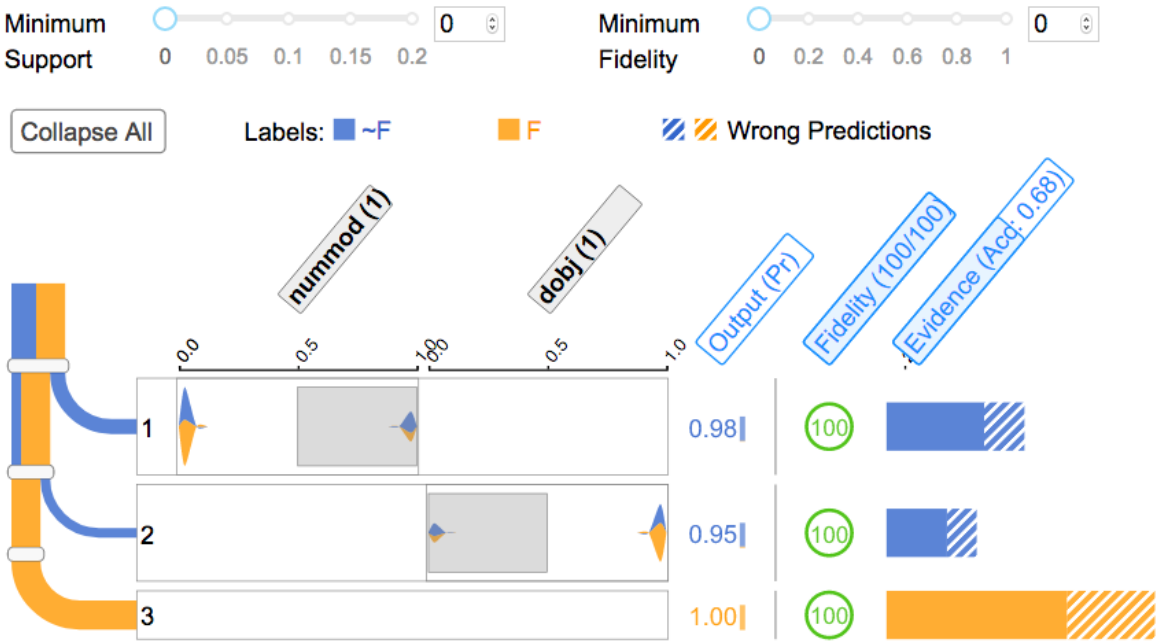


Figure 3.6. Interactive rule visualization interface of RuleMatrix.

3.5.3. Enhancing the Feature Selection with Features for Root Verbs

After observing the loss of performance with Q and OnlyQ, we combine the idea that uses keywords as features for non-functional requirement [4] with the dependency type *root*, the main verb of a sentence to improve the performance of our feature set for quality aspects. We identify two lists of root verbs that frequently occur in *OnlyF* and *OnlyQ* requirements: We select a list of root verbs in the data sets that occur 3 times more often in requirements tagged with quality aspects rather than in requirements tagged with functional aspects, and *vice versa*.

Table 3.9. Results with the features selected by Skoperules and Rulematrix.

Target	**Dataset**	**Prec**	**Rec**	**F1**	**AUC**
F	PROMISE	0.67	0.73	0.69	0.75
	Industry	0.83 ± 0.16	0.87 ± 0.08	0.8 ± 0.09	0.69 ± 0.09
Q	PROMISE	0.83	0.75	0.75	0.79
	Industry	0.57 ± 0.18	0.36 ± 0.14	0.65 ± 0.09	0.62 ± 0.05
OnlyF	PROMISE	0.71	0.64	0.69	0.74
	Industry	0.82 ± 0.19	0.78 ± 0.15	0.75 ± 0.1	0.67 ± 0.13
OnlyQ	PROMISE	0.84	0.64	0.71	0.78
	Industry	0.63 ± 0.2	0.29 ± 0.12	0.65 ± 0.12	0.58 ± 0.05

We also list verbs with *functional prevalence*, each occurring 3 times more often in OnlyF than in OnlyQ as well as in F than in Q; and similarly list verbs with *quality prevalence*. We set a minimum threshold of 10 occurrences in our data sets to exclude rare verbs.

Table 3.10 lists the final feature set using definitions adapted from the Universal Dependencies project [46], a worldwide attempt to reconcile the existing dependency parsing tag sets. Based on these verbs, listed in Table 3.10, we integrate the 15 most meaningful features from Section 3.5.2, with two additional boolean features *fverb* and *qverb*, which are true when a requirement contains one of the verbs in the list of most frequent verbs respectively in *OnlyF* and *OnlyQ* as a root verb, and we obtain a final feature set composed by 17 features. Table 3.11 reports the results obtained using such features for the classification.

Table 3.10. The final feature set.

Name	**Tag**	**Description**
Adjectival clause	acl	Clause that acts as an adjective and modifies a nominal.
Adverbial modifier	advmod	Adverb or adverbial phrase that modifies a predicate or a modifier word.
Adjectival modifier	amod	Adjectival phrase modifying a (pro)noun.
Passive auxiliary	auxpass	Non-main verb of the clause that contains passive information.
Direct object	dobj	The noun phrase that denotes the entity acted upon.
Nominal subject	nsubj	The nominal phrase which is the syntactic subject of a clause.
Nominal modifier	nmod	Noun acting as a non-core argument.
Numeric modifier	nummod	Number phrase that modifies the meaning of a noun with a quantity.
Passive nominal subject	nsubjpass	Noun phrase which is the syntactic subject of a passive clause.
Object of preposition	pobj	Link between a preposition and its object
Prepositional modifier	prep	A prepositional phrase that modifies the meaning of a verb, adjective, noun or another prep.
Adjectival complement	acomp	Number of adjectival phrases that function as complements of a verb (only root verbs included).
Cardinal	CD	Number of cardinal numbers (POS tag).
Modal	MD	Number of modal verbs (POS tag).
Adverb	RB	Number of adverbs (POS tag).

Table 3.10. The final feature set. (cont.)

Functional verb	fverb	Is the root verb one of {allow, display, send, track, include, notify, shall, add, assign, generate, request, create, define, record, indicate, save}?
Quality verb	qverb	Is the root verb one of {be, use, ensure, interface, handle, take, comply, run}?

While the results for F are comparable to those of Table 3.9, the additional features provide a slight improvement for OnlyQ and significantly better results for Q the recall on the industrial data sets increased from 0.36 to 0.70. Conversely, the features lead to a lower recall for OnlyF: -0.28. Further exploration is necessary to determine which are the appropriate linguistic features that denote quality aspects. Finally, comparing our results to the top-500 version of the high-dimensional classifier of Section 3.4, we observe that:

- On PROMISE, the results are slightly worse, but the degradation is limited (max -0.12);
- On the industry data sets, our approach shows substantial improvements in recall for F (+0.2), OnlyQ (+0.15), and especially OnlyF (+0.35).
- For the OnlyQ target, our approach is considerably worse in recall (-0.3), but shows a large gain in precision (+0.23).
- In our paper, we had used a subset of data sets, not all industrial data sets. Comparing the results with the paper, the results we obtained using the full data sets are worse. The performance of the newer industrial data sets turned out to have a much lower performance in F and Q both for our feature sets and our reference classifier so that the overall averages are lower (-0.8) compared the paper.

- The ROC plot of Figure 3.7 shows that, for the F case, a classifier with our features trained on PROMISE does not degrade on the industrial data sets. The plots for the other targets, instead, are similar to those in Figure 3.3.

Table 3.11. Results with additional features for root verbs.

Target	**Data Set**	**Prec**	**Rec**	**F1**	**AUC**
F	PROMISE	0.71	0.76	0.73	0.78
	Industry	0.82 ± 0.16	0.87 ± 0.09	0.8 ± 0.09	0.80 ± 0.09
Q	PROMISE	0.77	0.80	0.92	0.80
	Industry	0.55 ± 0.22	0.70 ± 0.11	0.65 ± 0.09	0.68 ± 0.07
OnlyF	PROMISE	0.77	0.83	0.72	0.82
	Industry	0.89 ± 0.16	0.50 ± 0.17	0.61 ± 0.11	0.80 ± 0.08
OnlyQ	PROMISE	0.71	0.75	0.78	0.64
	Industry	0.73 ± 0.16	0.32 ± 0.09	0.64 ± 0.11	0.66 ± 0.07

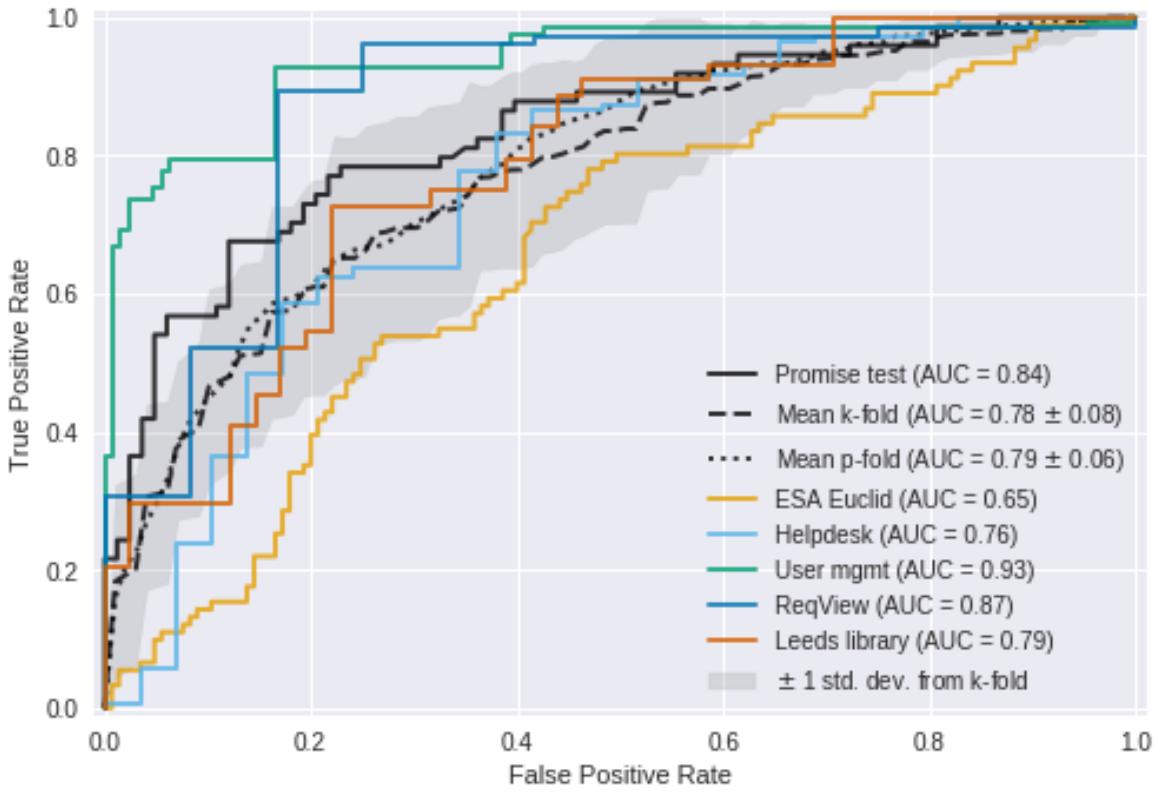


Figure 3.7. ROC plot for F with the final 17 features.

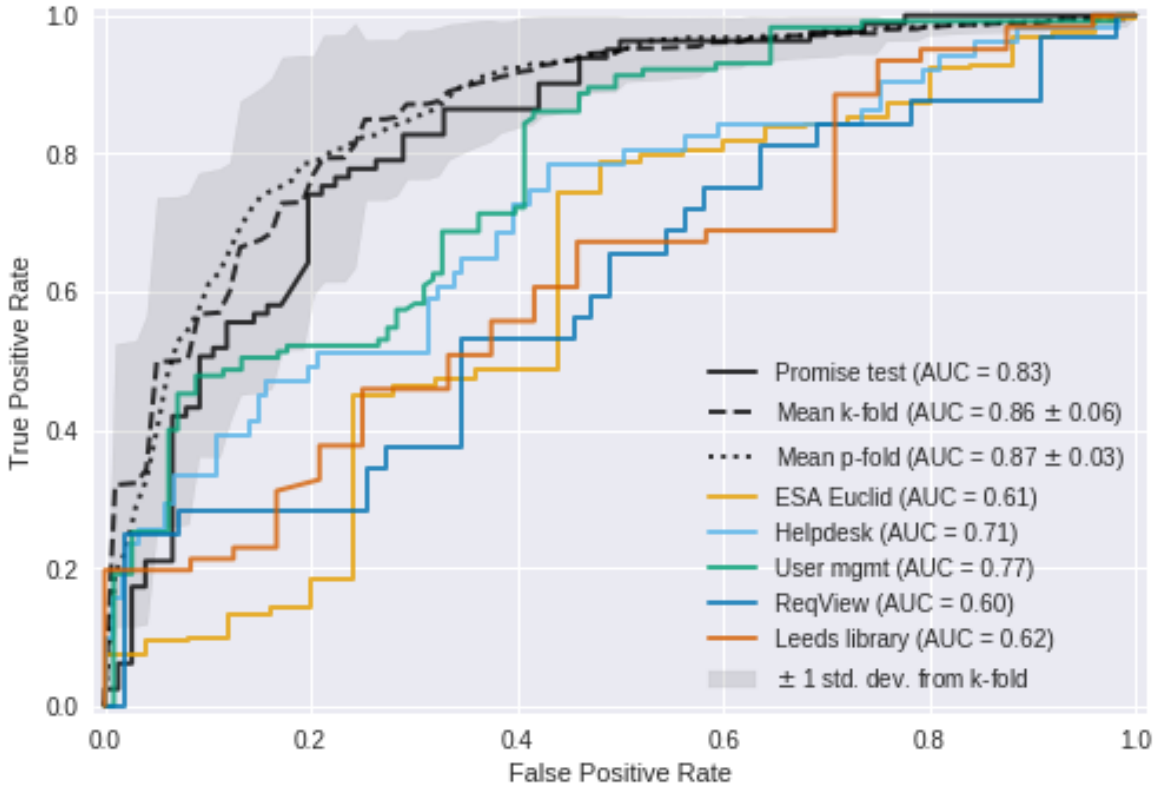


Figure 3.8. ROC plot for OnlyQ with the final 17 features.

3.5.4. Experimental Feature Selection with Automated Feature Selection

We apply additional experiments using automated feature selection algorithms to compare the performance of the automated feature selection algorithms and to see if we can improve the performance of our feature selection. As the most extended feature set with 75 features, we use Feature Set 3 (FS3) that we obtained in Section 3.5.1 as the basis to apply the automated feature selection algorithms. We select several algorithms using the known methods as well as some known phyton libraries, to see the applicability in industrial data sets. The source codes for our experiments can be found publicly in github [47]. We apply the following algorithms:

- *Forward Feature Selection*: This algorithm is a wrapper method which has a mining algorithm and uses the performance as its evaluation criteria to search and find the best feature [48].

  In the forward feature selection, the feature that performs the best is selected from all features which is tried in combination with all the other features. The number of features to be selected are defined at the beginning, so that the selection process continues until the specified number of features are selected. With this approach, the critical decision factor is to set the right number of target features to find the optimal feature set.

  In our experiments, we limited the number of features as 17 to compare the results set with the most successful feature set that we obtained in Section 3.5.3. Even with this limited feature set, the execution of the selection algorithm lasted around 4 hours for each F and Q, and the selected feature set was quite different from ours and the results were not impressive. After applying our classifier, the ROC Plot for F with the 17 Features selected using FFS from Feature Set 3 can be found in Figure 3.9 and Figure 3.10.

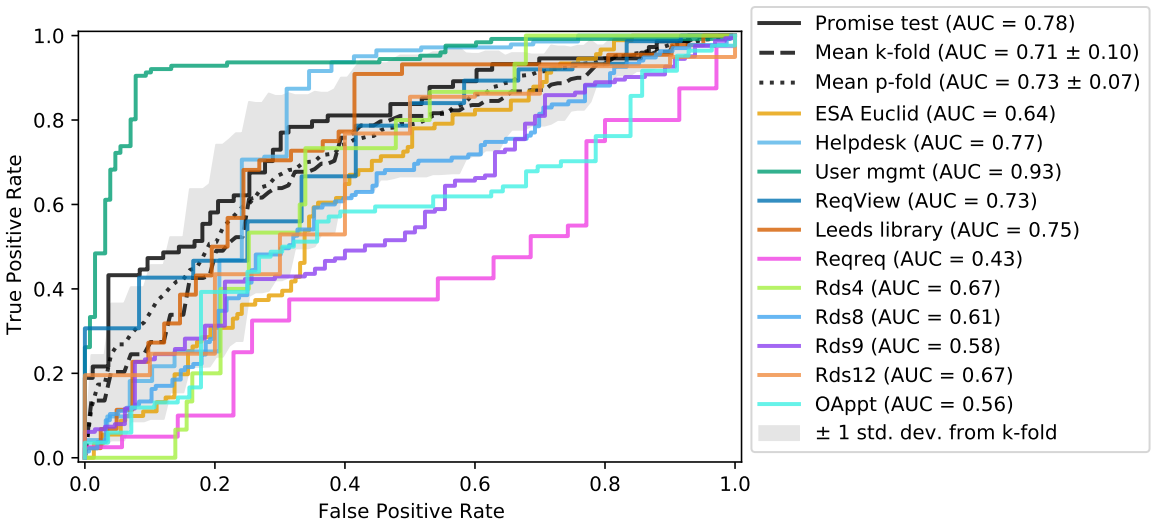


Figure 3.9. FFS - ROC Plot for F with selected features by FFS from FS 3.

- *Exhaustive Feature Selection*: Being another wrapper algorithm, the exhaustive feature selection evaluates the performance of the classifier against all possible combinations of the features in the data set.

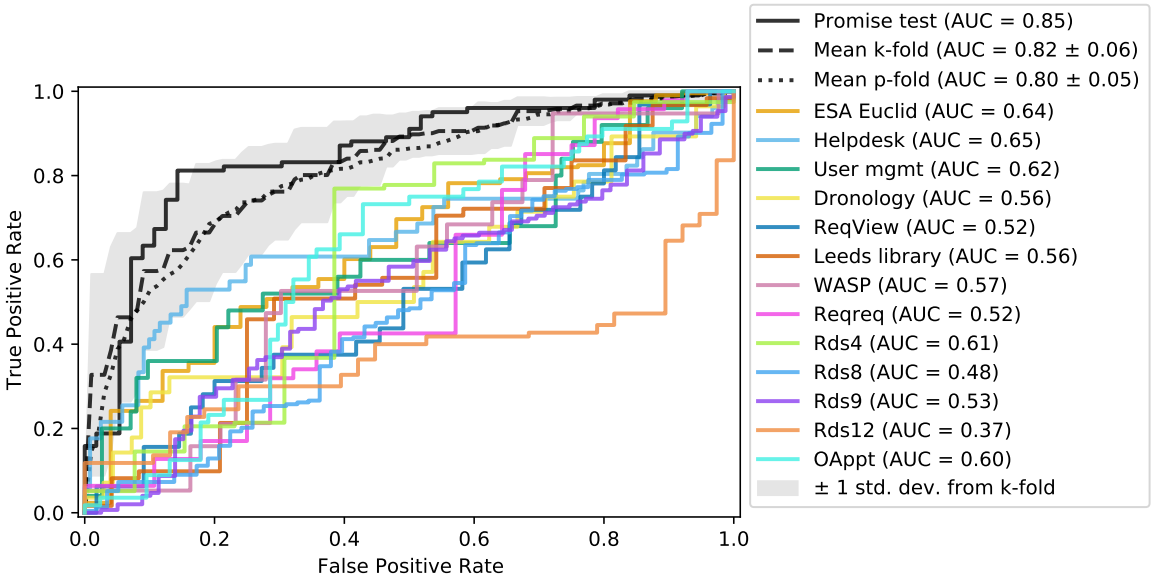


Figure 3.10. FFS - ROC Plot for Q with selected features by FFS from FS 3.

The feature subset that returns the best performance is selected. However, since it tries all the combinations of features and selects the best and therefore, the execution had performance problems. We wanted to select the best 17 features from the Feature Set 3, which had 75 features to compare the feature set and its results to our best selection, however, it was not possible to conclude the execution of the selection algorithm, even after running the algorithm in a few days.

- *Genetic Feature Selection*: Genetic algorithms are global optimization techniques for searching big data, like a sort of randomized search. They work throughout combinations of possible solutions, wherein each solution in the search space is represented as a finite length string over some finite set of symbols, which then uses an objective function to evaluate the suitability of each solution. In terms of feature selection, each string will represent a feature subset, and it will be represented with binary encoding: 1 means “choose” a given feature, and 0 means “do not choose” a feature. So for instance, the string 1001 means choose the first and the last feature as a feature subset [49].

Our genetic algorithm based on "Select from model" which is one of sklearn's built-in feature selection methods did not provide a satisfactory simplification in the number of features (53 features for Functional and 43 features for Non-functional requirements) nor in the results of the classifier. After applying our classifier, the ROC Plot for F with the 53 Features selected using Genetic Algorithm from Feature Set 3 can be found in Figure 3.11, and the ROC Plot for Q with the 53 Features selected using Genetic Algorithm from Feature Set 3 can be found in Figure 3.12.

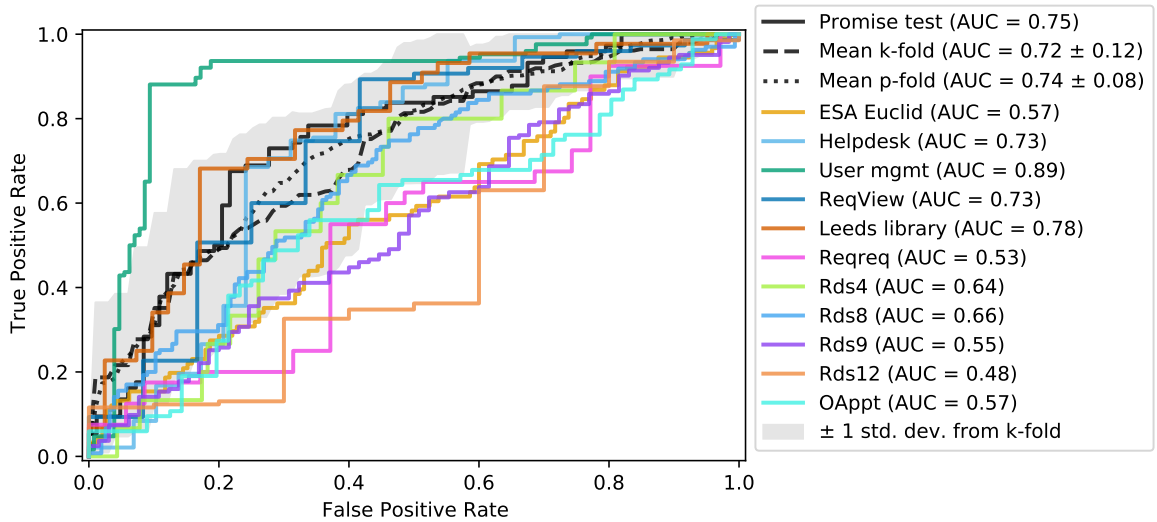


Figure 3.11. Genetic Algorithm - ROC Plot for F with selected features by GA from Feature Set 3.

- *Featurewiz*: Featurewiz is a python library to find the best features in a data set using the data frame and the name of the target feature. It removes highly correlated features automatically and then recursively does feature selection [50]. Featurewiz uses two back-to-back methods to remove any unnecessary features; SULOV (Searching for Uncorrelated List of Variables) and Recursive XGBoost. SULOV means Searching for Uncorrelated List of Variables. It finds all pairs of highly correlated variables exceeding a limit and assigns a score for that pair. After that, it removes the one with the lower score. Then, it uses the XGBoost to repeatedly find the best features among the remaining variables after SULOV.

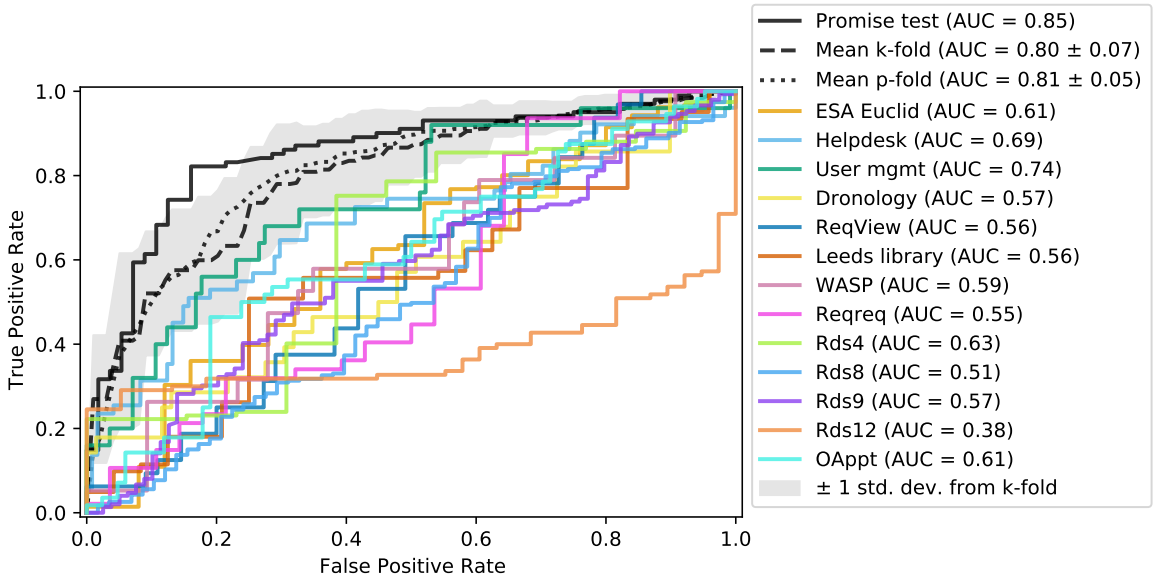


Figure 3.12. Genetic Algorithm - ROC Plot for Q w ith selected features by GA from Feature Set 3.

The Recursive XGBoost method selects all variables in the data set and the full data split into train and valid sets. It finds top X features on the train data set and repeats these multiple times to find the best features [51]. We have obtained the best results using Featurewiz, even better than our reference classifier as well as the final feature set we used in our paper. However, we realized that each time we execute the feature selection, Featurewiz returns different feature sets, which may end up with different results, therefore we concluded that this selection algorithm is not stable and reliable for now, and we decided to try a very similar but stable version, which is AutoVIML.

- *AutoVIML* Selection Algorithm: Auto-ViML is an open-source project that creates select the features, and builds a highly interpretable model based on the features [51]. AutoViML removes highly correlated variables and uses important features from XGBoost Tree Algorithm. Using this method, from 75 features, we ended up with 20 important functional features and 16 non-functional features. Figure 3.13 and Figure 3.14 display the importance level of respective features. After applying our classifier, the ROC Plot for F with the 20 Features selected using AutoVIML from Feature Set 3 can be found in Figure 3.15 and the ROC Plot for Q with the 16 Features from Feature Set 3 can be found in Figure 3.16.

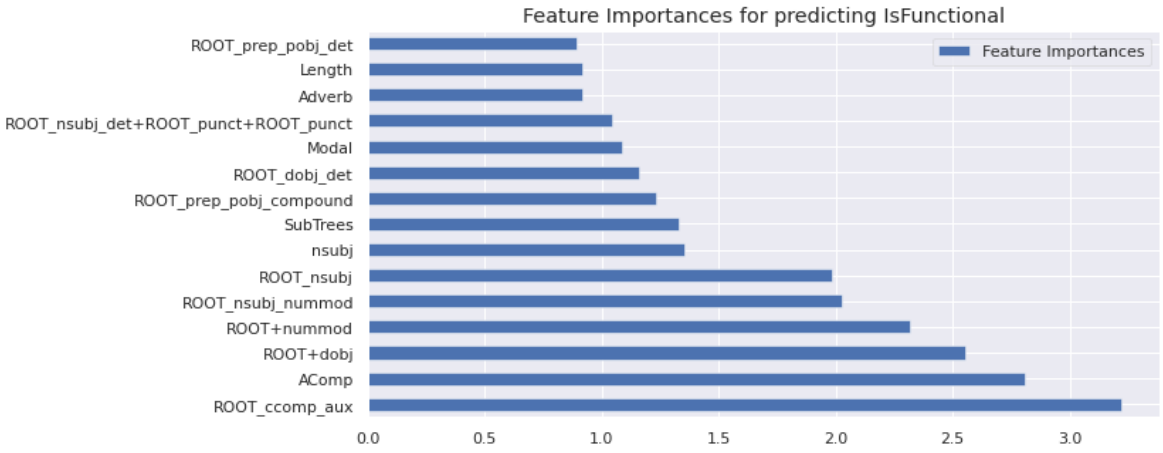


Figure 3.13. AutoVIML - Feature Importances for predicting F.

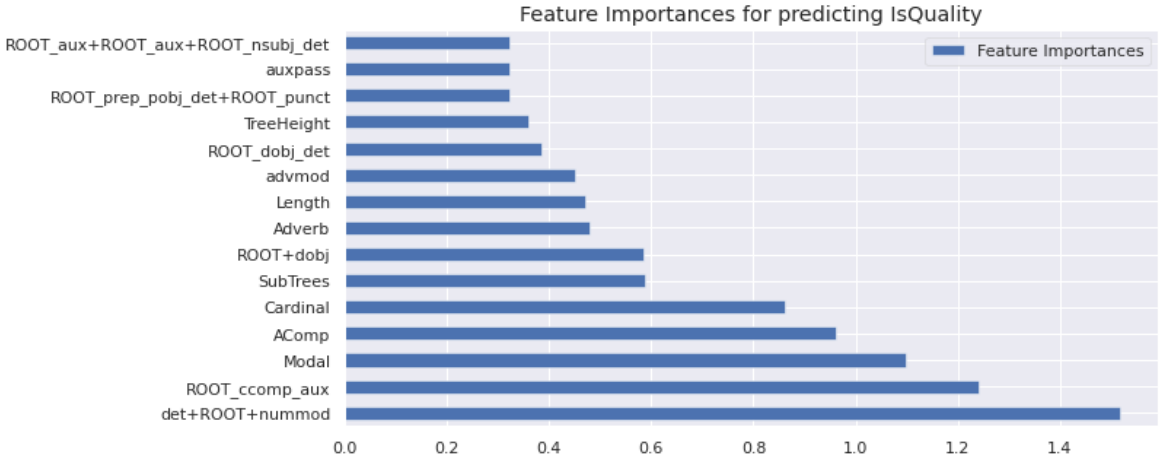


Figure 3.14. AutoVIML - Feature Importances for predicting Q.

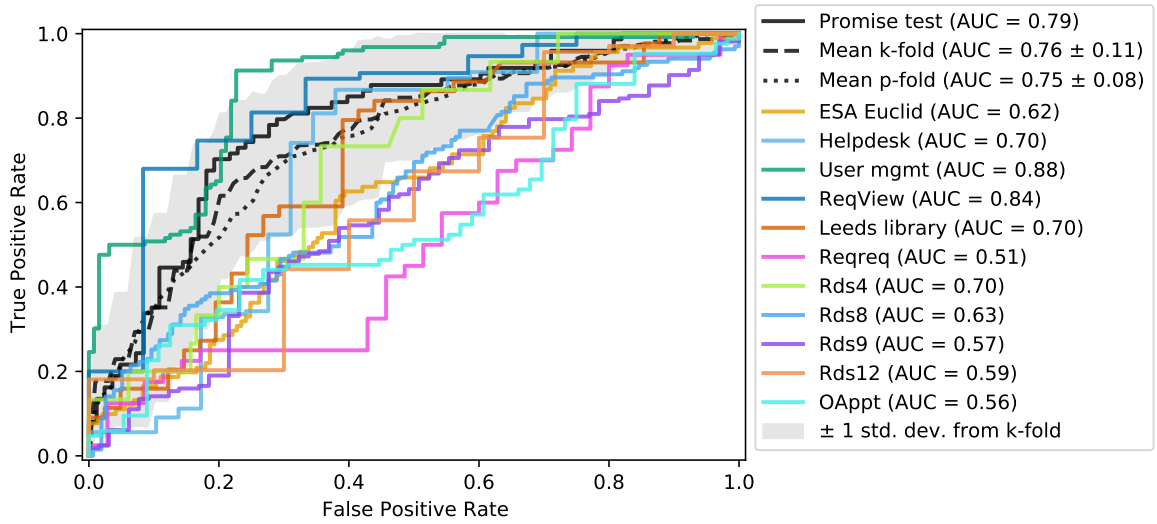


Figure 3.15. AutoVIML - ROC Plot for F with selected features by AutoVIML from Feature Set 3.

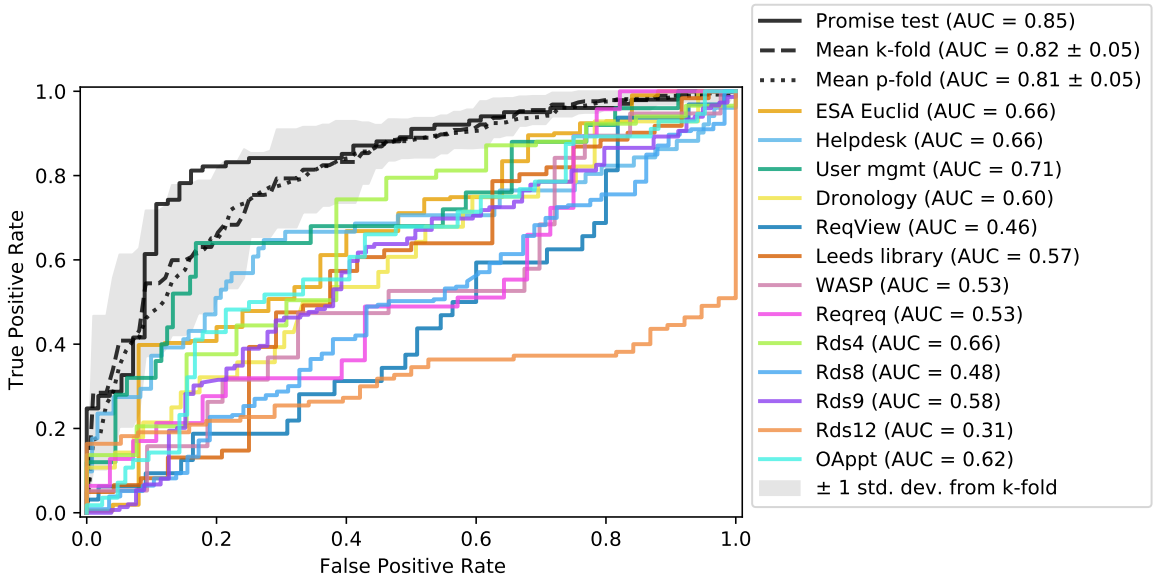


Figure 3.16. AutoVIML - ROC Plot for Q with selected features by AutoVIML from Feature Set 3.

- *Recursive Feature Selection*: The recursive feature elimination (RFE) selects features by grouping the features smaller and smaller recursively. The algorithm is trained on the initial set of features and each feature is assigned a value. This value is either through a coefficient attribute or through an importance attribute. Then, the least important features are removed from the current set of features. This process is recursively repeated on the feature set until the desired number of features to select is eventually reached [52]. This method returned the best result consistent feature set with 28 features for functional and 10 for non-functional requirements. Figure 3.17 and Figure 3.18 display the importance level of features.

  After applying our classifier, the ROC Plot for F with the 28 Features selected using Recursive Selection from Feature Set 3 can be found in Figure 3.19 and the ROC Plot for Q with the 10 Features selected using Recursive Selection from Feature Set 3 can be found in Figure 3.20.

- *A hybrid approach applying first Recursive then AutoVIML selection*: Since we did not achieve a big simplification in the number of features using the Recursive Feature Selection, we decided to try combinations. First, we chose to apply AutoVIML selection algorithm to the feature set selected by Recursive Selection.

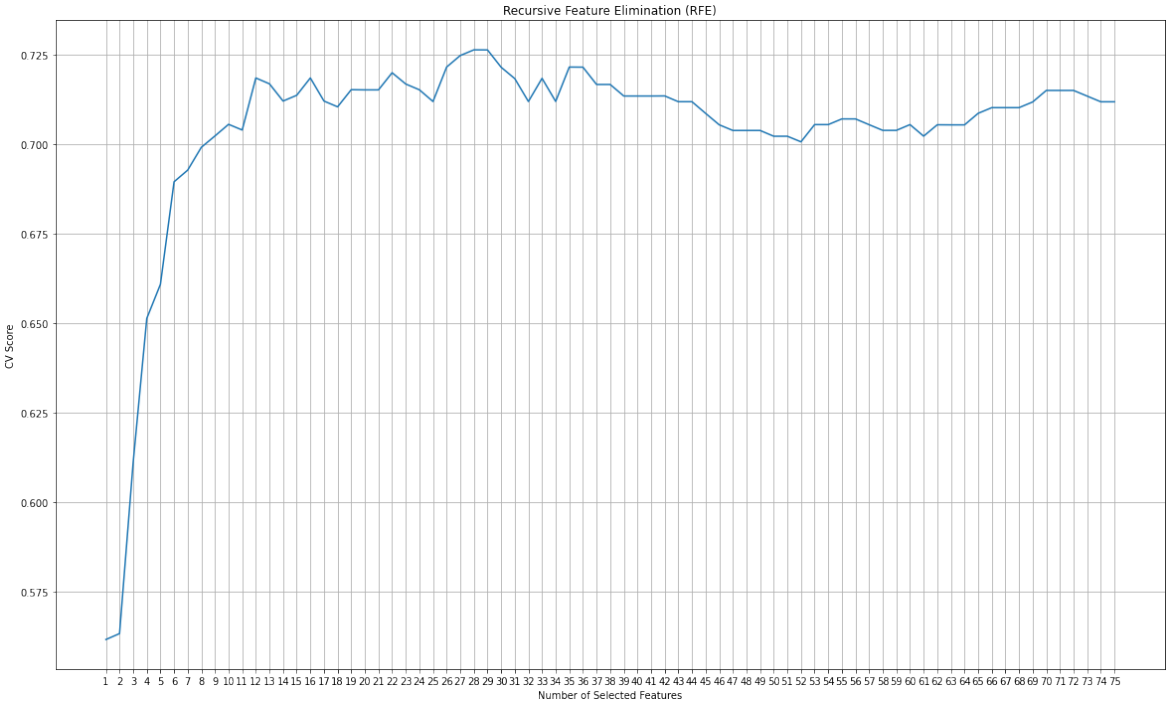


Figure 3.17. RFE - Feature Importances for predicting F.

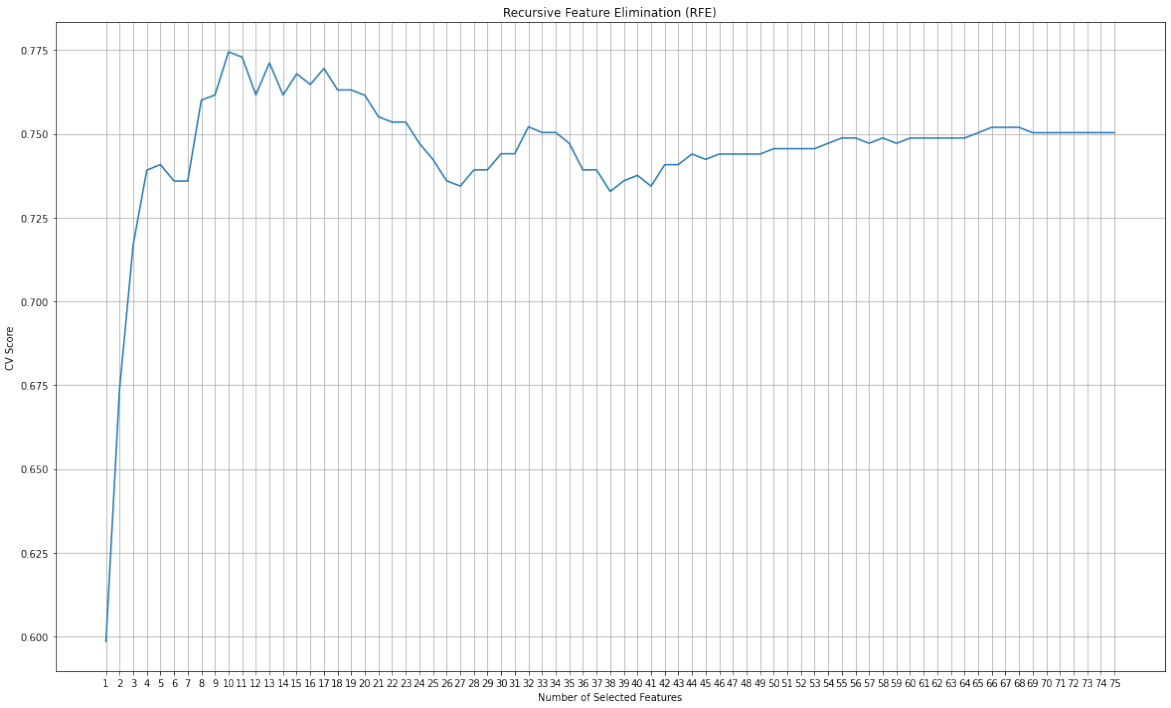


Figure 3.18. RFE - Feature Importances for predicting Q.

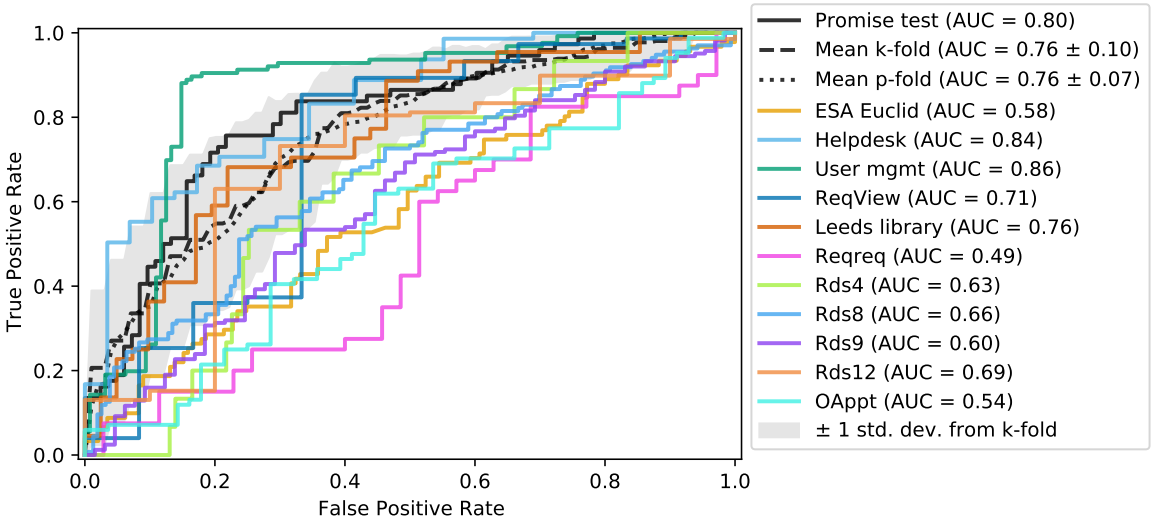


Figure 3.19. Recursive Feature Selection - ROC Plot for F with selected features by RFE from Feature Set 3.

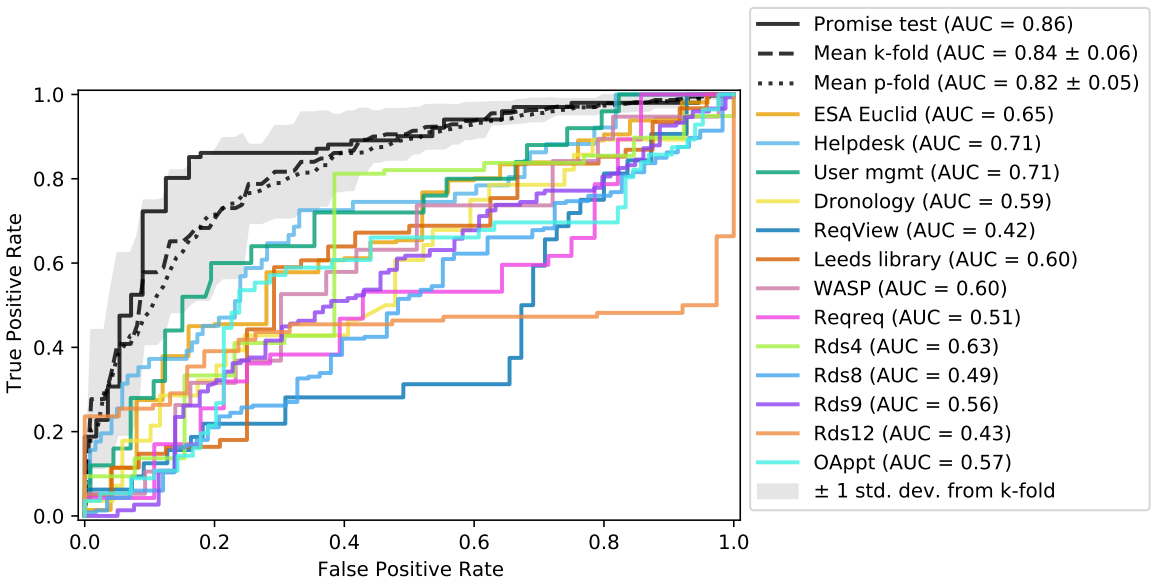


Figure 3.20. Recursive Feature Selection - ROC Plot for Q with selected features by RFE from Feature Set 3.

This experiment resulted in 14 features, however, the results were slightly worse than the Recursive Selection.

- *A hybrid approach applying first AutoVIML then Recursive selection*: As the next attempt, we applied the recursive selection to the 20 features that were selected with AutoVIML, which ended with 17 features and the results were disappointing.

The results of our experiments can be found in Table 3.12 and Table 3.13. Due to space constraints, we do not list the features selected by each algorithm, however, all features selected can be found in B and C. Using the automated feature selection, although we have achieved a simplification of the feature set, the features selected by each experiment was completely different from each other, and the results did not bring any improvement compared to the results we obtained using the feature set we chose using interpretable machine learning components.

Table 3.12. AUC Comparison of all feature sets for F classifier target.

Feature Selection	**# Feat.**	**Promise**	**Euclid**	**Helpdesk**	**User mgmt**	**Dronology**	**Reqview**
Kurtanovic Top 100	100	0.90	0.66	0.87	0.85	0.97	0.86
Kurtanovic Top 500	500	0.89	0.59	0.78	0.67	0.77	0.66
FS1	17	0.73	0.67	0.78	0.95	0.31	0.60
FS2	27	0.68	0.62	0.80	0.92	0.70	0.52
FS3	75	0.76	0.63	0.80	0.89	0.55	0.80
FS3 with Interpretable ML	15	0.75	0.63	0.68	0.91	0.30	0.77
FS3 with Interpretable ML + verbs	17	0.84	0.63	0.78	0.93	0.66	0.82
FS3 with Recursive Feature Elimination	28	0.80	0.58	0.84	0.86	0.53	0.71
FS3 + Featurewiz	15	0.89	0.69	0.94	0.99	0.59	0.95
FS3 + Featurewiz 2nd try	17	0.89	0.71	0.93	0.99	0.70	0.89
FS3 + Forward Feature Selection	17	0.78	0.64	0.78	0.93	0.42	0.73
FS3+ Genetic Search Algorithm	53	0.87	0.66	0.71	0.71	0.59	0.42
FS3 with AutoVIML and Recurs.FS	17	0.85	0.58	0.64	0.74	0.28	0.81
FS3 with Recurs.FS and AutoVIML	14	0.78	0.65	0.77	0.90	0.71	0.79
FS3 with AutoVIML	20	0.79	0.62	0.70	0.88	0.47	0.84

Table 3.12. AUC Comparison of all feature sets for F classifier target. (cont.)

Feature Selection	**# Feat.**	**Leeds**	**WASP**	**Repreq**	**Rds4**	**Rds8**	**Rds9**
Kurtanovic Top 100	100	0.73	0.64	0.61	0.75	0.68	0.73
Kurtanovic Top 500	500	0.72	0.66	0.69	0.68	0.62	0.63
FS1	17	0.68	0.63	0.50	0.75	0.57	0.60
FS2	27	0.62	0.55	0.58	0.58	0.58	0.59
FS3	75	0.78	0.85	0.52	0.65	0.65	0.61
FS3 with Interpretable ML	15	0.75	0.66	0.46	0.66	0.61	0.58
FS3 with Interpretable ML + verbs	17	0.75	0.86	0.60	0.77	0.65	0.64
FS3 with Recursive Feature Elimination	28	0.76	0.80	0.49	0.63	0.66	0.60
FS3 + Featurewiz	15	0.82	0.82	0.65	0.64	0.67	0.58
FS3 + Featurewiz 2nd try	17	0.85	0.93	0.62	0.67	0.75	0.61
FS3 + Forward Feature Selection	17	0.75	0.78	0.43	0.67	0.62	0.58
FS3+ Genetic Search Algorithm	53	0.60	0.60	0.51	0.63	0.49	0.56
FS3 with AutoVIML and Recurs.FS	17	0.72	0.86	0.62	0.53	0.34	0.45
FS3 with Recurs.FS and AutoVIML	14	0.66	0.75	0.55	0.75	0.64	0.57
FS3 with AutoVIML	20	0.70	0.75	0.51	0.70	0.63	0.57

Table 3.12. AUC Comparison of all feature sets for F classifier target. (cont.)

Feature Selection	**# Feat.**	**Rds12**	**OAppt**	**Ind.Avg.**	**Std-dev**		
Kurtanovic Top 100	100	0.71	0.64	0.75	0.11		
Kurtanovic Top 500	500	0.69	0.58	0.67	0.06		
FS1	17	0.50	0.60	0.63	0.15		
FS2	27	0.45	0.62	0.62	0.12		
FS3	75	0.60	0.57	0.69	0.12		
FS3 with Interpretable ML	15	0.49	0.65	0.63	0.15		
FS3 with Interpretable ML + verbs	17	0.70	0.74	0.73	0.09		
FS3 with Recursive Feature Elimination	28	0.69	0.54	0.67	0.12		
FS3 + Featurewiz	15	0.58	0.57	0.73	0.15		
FS3 + Featurewiz 2nd try	17	0.71	0.62	0.77	0.13		
FS3 + Forward Feature Selection	17	0.67	0.56	0.66	0.14		
FS3+ Genetic Search Algorithm	53	0.43	0.57	0.58	0.09		
FS3 with AutoVIML and Recurs.FS	17	0.40	0.53	0.58	0.17		
FS3 with Recurs.FS and AutoVIML	14	0.59	0.60	0.69	0.10		
FS3 with AutoVIML	20	0.60	0.56	0.66	0.12		

Table 3.13. AUC Comparison of all feature sets for Q classifier target.

Feature Selection	**# Feat.**	**Promise**	**Euclid**	**Helpdesk**	**User mgmt**	**Dronology**	**Reqview**
Kurtanovic Top 100	100	0.948	0.593	0.736	0.575	0.71	0.611
Kurtanovic Top 500	500	0.917	0.537	0.673	0.472	0.657	0.584
FS1	17	0.775	0.636	0.632	0.728	0.598	0.495
FS2	27	0.625	0.553	0.626	0.579	0.619	0.568
FS3	75	0.839	0.624	0.733	0.714	0.592	0.49
FS3 with Interpretable ML	15	0.803	0.616	0.671	0.705	0.578	0.486
FS3 with Interpretable ML + verbs	17	0.877	0.614	0.739	0.761	0.749	0.641
FS3 with Recursive Feature Elimination	10	0.864	0.654	0.708	0.712	0.586	0.424
FS3 + Featurewiz	14	0.827	0.685	0.657	0.713	0.622	0.476
FS3 + Forward Feature Selection	17	0.849	0.642	0.654	0.617	0.556	0.519
FS3+ Genetic Search Algorithm	42	0.852	0.613	0.691	0.738	0.568	0.564
FS3 with AutoVIML and Recurs.FS	7	0.829	0.68	0.63	0.732	0.602	0.468
FS3 with Recurs.FS and AutoVIML	12	0.835	0.661	0.647	0.692	0.572	0.575
FS3 with AutoVIML	16	0.846	0.663	0.659	0.709	0.596	0.455

Table 3.13. AUC Comparison of all feature sets for Q classifier target. (cont.)

Feature Selection	**# Feat.**	**Leeds**	**WASP**	**Repreq**	**Rds4**	**Rds8**	**Rds9**
Kurtanovic Top 100	100	0.449	0.603	0.587	0.7	0.634	0.588
Kurtanovic Top 500	500	0.523	0.613	0.554	0.68	0.637	0.501
FS1	17	0.638	0.536	0.457	0.597	0.551	0.629
FS2	27	0.551	0.421	0.493	0.472	0.636	0.642
FS3	75	0.609	0.565	0.568	0.625	0.527	0.574
FS3 with Interpretable ML	15	0.627	0.558	0.492	0.643	0.477	0.579
FS3 with Interpretable ML + verbs	17	0.641	0.581	0.489	0.652	0.488	0.583
FS3 with Recursive Feature Elimination	10	0.6	0.597	0.513	0.633	0.489	0.556
FS3 + Featurewiz	14	0.592	0.559	0.536	0.635	0.472	0.583
FS3 + Forward Feature Selection	17	0.558	0.57	0.524	0.613	0.478	0.531
FS3+ Genetic Search Algorithm	42	0.561	0.591	0.551	0.63	0.51	0.571
FS3 with AutoVIML and Recurs.FS	7	0.628	0.602	0.527	0.631	0.495	0.543
FS3 with Recurs.FS and AutoVIML	12	0.516	0.541	0.54	0.655	0.517	0.588
FS3 with AutoVIML	16	0.569	0.528	0.528	0.661	0.483	0.578

Table 3.13. AUC Comparison of all feature sets for Q classifier target. (cont.)

Feature Selection	**# Feat.**	**Rds12**	**OAppt**	**Ind.Avg.**	**Std-dev**		
Kurtanovic Top 100	100	0.523	0.566	0.606	0.075		
Kurtanovic Top 500	500	0.717	0.684	0.602	0.076		
FS1	17	0.664	0.573	0.595	0.07		
FS2	27	0.348	0.59	0.546	0.086		
FS3	75	0.385	0.556	0.582	0.086		
FS3 with Interpretable ML	15	0.561	0.628	0.586	0.069		
FS3 with Interpretable ML + verbs	17	0.368	0.622	0.61	0.108		
FS3 with Recursive Feature Elimination	10	0.432	0.57	0.575	0.089		
FS3 + Featurewiz	14	0.37	0.612	0.578	0.092		
FS3 + Forward Feature Selection	17	0.373	0.604	0.557	0.073		
FS3+ Genetic Search Algorithm	42	0.38	0.611	0.583	0.082		
FS3 with AutoVIML and Recurs.FS	7	0.418	0.567	0.579	0.084		
FS3 with Recurs.FS and AutoVIML	12	0.402	0.618	0.579	0.075		
FS3 with AutoVIML	16	0.315	0.618	0.566	0.102		

3.6. Threats to Validity

We have spotted several challenges and threats during this study.

Internal validity: Tagging of requirements as Q and F is a tagger-biased objective evaluation. Based on their experiences and knowledge, taggers may tag the same requirement differently. Although we mitigated this through the tagging reconciliation meetings, yet it is still highly possible that other taggers with different backgrounds may have produced different tagging results with different standards, and that the results may have been slightly different.

Construct validity: Despite our thorough efforts and our attempt to obtain the source code of [1], we failed, and we had to reconstruct our reference classifier, however, our reconstruction of their feature set is not perfect. The analysis of the most informative features, however, reveals a high degree of similarity. Also, it is possible that we could have omitted some elements of [40] during the reconstruction process.

External validity: Our attempt in identifying requirements data sets that represent heterogeneous industrial practices led to a varying performance across the data sets. Our feature set is the basis for constructing classifiers that possess sufficient initial performance, but domain adaptation is still needed. To mitigate this possible threat to the generalization, we publicly share our code and data for further replications and studies.

Conclusion validity: Unbalanced data sets are data sets where all classes are not evenly represented. This is a general problem in the RE field: with large-scale data sets, it is not possible to use mitigation techniques like under-sampling. This situation is a great risk for the validity of the statistical results.

4. CONCLUSIONS

This study explores interpretable ML as a tool to build and evaluate classifiers in RE by investigating the well-known problem of distinguishing functional and quality aspects in requirements collections. Through this approach, we build a feature set that is applied to an ML classifier. The quantitative and qualitative results show that our feature set, largely based on linguistic dependencies, achieves similar performance to high-dimensional feature sets with lower-level feature types (e.g., text n-grams and POS n-grams). On the other hand, the experiments to further optimize the feature set with automated feature selection algorithms did not return any success. This is mainly because of the reason that automated feature selection algorithms try to find the statistically best relationships between features and feature combinations, but do not take the linguistic dependencies into account.

For interpretable ML to be effective, it is important to rely on a limited set of features that have clear semantics. We rely on linguistic dependencies that define the main relationships in a sentence as opposed to the low-level short sequences of words (n-grams) used by other researchers [1]. The limited number of features (only 17) facilitates the analysis of the inner working of a classifier. Nevertheless, the results also show that identifying features that denote quality aspects is more difficult than determining those that denote functional aspects. This feature set that we put forward, alongside our followed design process that is assisted by interpretable ML, constitutes the basis for the construction of new requirements classifiers that rely on higher-level linguistic features.

Future work should focus in improvements on constructing classifiers. While we do not prescribe a single way of using our feature set, some possible ways to use our results are the following:

- Since the root verb types features degrade the performance for OnlyF, one could use an earlier feature set for that classification task, leading to a significant gain in recall;
- The binary classifier can be turned into a recommendation tool that provides degrees of membership for the various quality aspects: if we take F and Q, the probability that a classifier assigns can be kept explicit instead of using it as a yes/no threshold. For example, a requirement could have 90% likelihood to have functional aspects, and 60% likelihood to have quality aspects.
- The list of functional and quality verbs can be customized for the domain of use, for certain qualities may be more prevalent in a domain rather than in another.
- The interpretable ML techniques we employ can be applied to introspect the classifier's inner workings as it is being used, aiming at further improving the feature set.

REFERENCES

1. Kurtanović, Z. and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning", *IEEE International Requirements Engineering Conference (RE)*, pp. 490–495, 2017.

2. Minhas, N. M., S. Majeed, Z. Qayyum and M. Aasem, "Controlled vocabulary based software requirements classification", *2011 Malaysian Conference in Software Engineering*, pp. 31–36, IEEE, 2011.

3. Glinz, M., "A glossary of requirements engineering terminology", *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam*, Vol. 1, p. 56, 2011.

4. Cleland-Huang, J., R. Settimi, X. Zou and P. Solc, "Automated classification of non-functional requirements", *Requirements Engineering*, Vol. 12, No. 2, pp. 103–120, 2007.

5. Binkhonain, M. and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements", *Expert Systems with Applications*, 2019.

6. Abad, Z. S. H., O. Karras, P. Ghazi, M. Glinz, G. Ruhe and K. Schneider, "What works better? A study of classifying requirements", *IEEE International Requirements Engineering Conference (RE)*, pp. 496–501, 2017.

7. Maalej, W., Z. Kurtanović, H. Nabil and C. Stanik, "On the automatic classification of app reviews", *Requirements Engineering*, Vol. 21, No. 3, pp. 311–331, 2016.

8. Winkler, J. and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks", *IEEE International Requirements Engineering*

Conference Workshops (REW), pp. 39–45, 2016.

9. Glinz, M., "On non-functional requirements", *IEEE International Requirements Engineering Conference (RE)*, pp. 21–26, 2007.

10. Gheyas, I. A. and L. S. Smith, "Feature subset selection in large dimensionality domains", *Pattern Recognition*, Vol. 43, No. 1, pp. 5–13, 2010.

11. Li, F.-L., J. Horkoff, J. Mylopoulos, R. S. Guizzardi, G. Guizzardi, A. Borgida and L. Liu, "Non-functional requirements as qualities, with a spice of ontology", *IEEE International Requirements Engineering Conference (RE)*, pp. 293–302, 2014.

12. Kübler, S., R. McDonald and J. Nivre, *Dependency parsing*, Morgan & Claypool Publishers, 2009.

13. Ming, Y., H. Qu and E. Bertini, "RuleMatrix: Visualizing and understanding classifiers with rules", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 25, No. 1, pp. 342–352, 2019.

14. Dalpiaz, F., D. Dell'Anna, F. B. Aydemir and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing", *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pp. 142–152, 2019.

15. Hey, T., J. Keim, A. Koziolek and W. F. Tichy, "NoRBERT: Transfer learning for requirements classification", *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 169–179, 2020.

16. "The PROMISE repository of empirical software engineering data", `https://terapromise.csc.ncsu.edu/!/repo/view/head/requirements/nfr`, [accessed 8-April-2019].

17. Casamayor, A., D. Godoy and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach", *Information*

and Software Technology, Vol. 52, No. 4, pp. 436–445, 2010.

18. Mahmoud, A., "An information theoretic approach for extracting and tracing non-functional requirements", *IEEE International Requirements Engineering Conference (RE)*, pp. 36–45, 2015.

19. Zhang, W., Y. Yang, Q. Wang and F. Shu, "An empirical study on classification of non-functional requirements", *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pp. 190–195, 2011.

20. Hussain, I., L. Kosseim and O. Ormandjieva, "Using linguistic knowledge to classify non-functional requirements in SRS documents", *International Conference on Application of Natural Language to Information Systems (NLDB)*, pp. 287–298, 2008.

21. Singh, P., D. Singh and A. Sharma, "Rule-based system for automated classification of non-functional requirements from requirement specifications", *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 620–626, 2016.

22. Navarro-Almanza, R., R. Juarez-Ramirez and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification", *International Conference in Software Engineering Research and Innovation (CONISOFT)*, pp. 116–120, 2017.

23. Kira, K. and L. A. Rendell, "A Practical approach to feature selection", *Machine Learning Proceedings 1992*, pp. 249–256, Morgan Kaufmann, San Francisco (CA), 1992.

24. Tabakhi, S., P. Moradi and F. Akhlaghian, "An unsupervised feature selection algorithm based on ant colony optimization", *Engineering Applications of Artificial Intelligence*, Vol. 32, pp. 112–123, 2014.

25. Liu, H. and L. Yu, "Toward integrating feature selection algorithms for classification and clustering", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 4, pp. 491–502, 2005.

26. Khurana, U., D. Turaga, H. Samulowitz and S. Parthasrathy, "Cognito: Automated feature engineering for supervised learning", *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pp. 1304–1307, 2016.

27. Boetticher, G., T. Menzies and T. Ostrand, "The PROMISE repository of empirical software engineering data", *Department of Computer Science*, Vol. 52, 2007, `https://terapromise.csc.ncsu.edu/!/repo/view/head/requirements/nfr`.

28. "ESA Euclide mission system requirements", `http://sci.esa.int/euclid/`, [accessed 8-April-2019].

29. Cleland-Huang, J., M. Vierhauser and S. Bayley, "Dronology: An incubator for cyber-physical systems research", *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp. 109–112, 2018.

30. "ReqView example requirements", `https://www.reqview.com/doc/example requirements-documents.html`, [Online; accessed 8-April-2019].

31. "Leeds University library requirements", `https://leedsunilibrary.files. .com/2013/06/repositoryfunctionalrequirementsv1-1`$_web_{1.}xlsx$.

32. "Web architectures for services platforms (WASP) requirements", `https://www.zenodo.org/record/581655`, [Online; accessed 8-April-2019].

33. "Non-functional requirements", `https://requirementsquest.com/wp-content uploads/2017/01/Nonfunctional-Requirement-EXAMPLES.pdf`, [accessed 8-December-2019].

34. Miller, R. E., *The quest for software requirements*, MavenMark Books,

Oconomowoc, WI, USA, 2009.

35. "System requirements: NATO CSD ISR streaming services (CISS)", `https://mpit.bip.gov.pl/fobjects`, [accessed 8-December-2019].

36. "Functional requirements for electronic records management systems", `https://www.nationalarchives.gov.uk/documents/requirements.pdf`, [accessed 8-December-2019].

37. "Michigan Department of Transportation's (MDOT's) vehicle infrastructure integration (VII) data use analysis and processing (DUAP) system.", `http://fmt.isti.cnr.it/nlreqdataset/SRSComparison`, [accessed 8-December-2019].

38. Fawcett, T., "An introduction to ROC analysis", *Pattern Recognition Letters*, Vol. 27, No. 8, pp. 861–874, 2006.

39. Hanley, J. A. and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve", *Radiology*, Vol. 143, No. 1, pp. 29–36, 1982.

40. Kurtanović, Z. and W. Maalej, "On user rationale in software engineering", *Requirements Engineering*, Vol. 23, No. 3, pp. 357–379, 2018.

41. MAST, "Applied software technology: App and user review analysis", `https://mast.informatik.uni-hamburg.de/app-review-analysis/`, [accessed 8-April-2019].

42. Kitaev, N. and D. Klein, "Constituency parsing with a self-attentive encoder", *Annual Meeting of the Association for Computational Linguistics (ACL): Volume 1, Long Papers*, 2018.

43. Anonymous, "Supplementary material for the paper: Featurewiz", `https://github.com/AutoViML/featurewiz`.

44. Gunning, D., "Explainable artificial intelligence (XAI)", https://www.darpa.mil/attachments/XAIProgramUpdate.pdf, 2017, [Online; accessed 8-April-2019].

45. "SkopeRules", https://skope-rules.readthedocs.io/en/latest/, [accessed 8-April-2019].

46. "Universal dependencies", http://universaldependencies.org/, [accessed 8-April-2019].

47. *Github source codes for automated feature selection algorithm experiments*, https://github.com/cevikol/SCO690_Thesis_Repository.

48. Visalakshi, S. and V. Radha, "A literature review of feature selection techniques and applications: Review of feature selection in data mining", *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1–6, 2014.

49. Madhu, G. and K. Reddy, "Data mining for genetics: A genetic algorithm approach", *J. Convergence Inf. Technol.*, Vol. 3, No. 3, pp. 39–45, 2008, http://www.aicit.org/jcit/ppl/jcit_vol3no3_6.pdf.

50. "Featurewiz", https://towardsdatascience.com/featurewiz-fast-way-to-select-best-features-in-a-data-9c8611, [accessed 8-April-2021].

51. "Featurewiz", http://universaldependencies.org/, [accessed 8-April-2021].

52. "sklearn.feature selection RFE", https://scikit-learn.org/stable/modules/generated/sklearn.featureselection.RFE.html, [accessed 8-April-2021].

APPENDIX A: FEATURE SETS

ID	Feature Set	# of F Features	# of Q Features
FS1	Feature Set 1	17	17
FS2	Feature Set 2	27	27
FS3	Feature Set 3	75	75
FS4	FS3 enhanced with Interpretable ML	15	15
FS5	FS4 enhanced with verbs	17	17
FS6	FS3 with Recursive Feature Elimination	28	10
FS7	FS3 with Featurewiz	17	14
FS8	FS3 with Forward Feature Selection	17	17
FS9	FS3 with Genetic Search Algorithm	53	42
FS10	FS3 with AutoVIML	20	16
FS11	FS3 with AutoVIML and Recursive Feature Elimination	17	7
FS12	FS3 with Recursive Feature Elimination and AutoVIML	14	12

APPENDIX B: FUNCTIONAL FEATURES PER FEATURE SET

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
`acl`	✓	✓	✓	✓	✓			✓	✓			
`AComp`		✓	✓	✓	✓	✓	✓		✓	✓	✓	
`Adjective`		✓	✓						✓			✓
`advcl`	✓		✓						✓			
`Adverb`		✓	✓	✓	✓		✓			✓	✓	✓
`advmod`	✓		✓							✓		✓
`AdvMod`	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
`amod`	✓		✓					✓				
`AMod`	✓	✓	✓	✓	✓			✓	✓			
`aux`	✓	✓										
`aux+det+dobj`			✓			✓						
`aux+dobj`			✓						✓			
`aux+nummod`			✓						✓			
`aux+nummod+punct`			✓						✓	✓		
`aux+ROOT+nummod`			✓						✓			
`auxpass`	✓		✓	✓	✓	✓		✓				✓

Table B. Functional Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
Cardinal		✓	✓	✓	✓	✓			✓		✓	
CompSupAdj		✓	✓				✓					
CompSupAdv		✓	✓			✓		✓	✓			
det	✓	✓										
det+aux+nummod			✓						✓		✓	
det+nummod			✓					✓	✓			
det+nummod+punct			✓						✓			
det+ROOT+nummod			✓						✓			
dobj	✓	✓	✓	✓	✓		✓	✓	✓			
dobj+pobj			✓						✓			
DTreeHeight		✓	✓					✓	✓			
hasFverb					✓					✓		✓
hasQverb					✓					✓		✓
Length		✓	✓			✓		✓	✓		✓	
Modal		✓	✓	✓	✓	✓	✓		✓		✓	
nmod	✓		✓	✓	✓			✓		✓		
Noun		✓	✓								✓	

Table B. Functional Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
nsubj	✓	✓	✓	✓	✓			✓	✓		✓	
nsubj+aux+dobj			✓						✓			
nsubj+det+dobj			✓			✓			✓			
nsubj+dobj			✓			✓		✓	✓		✓	
nsubj+nummod			✓						✓			
nsubjpass	✓		✓	✓	✓	✓	✓	✓	✓			
nummod	✓		✓	✓	✓			✓	✓			
nummod+pobj			✓			✓			✓			
nummod+punct			✓						✓			
pobj	✓	✓		✓	✓							
prep	✓	✓		✓	✓							
punct	✓	✓										
ROOT_aux		✓										
ROOT_aux+ROOT_aux			✓						✓			
ROOT_aux+ROOT_aux+ROOT_nsubj_det			✓			✓			✓		✓	
ROOT_aux+ROOT_aux+ROOT_punct			✓			✓			✓			
ROOT_aux+ROOT_dobj_acl_aux			✓			✓			✓			

Table B. Functional Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
ROOT_aux+ROOT_dobj_acl_aux+ROOT_dobj_det			✓			✓						
ROOT_aux+ROOT_dobj_acl_aux+ROOT_punct			✓						✓			
ROOT_aux+ROOT_dobj_det			✓									
ROOT_aux+ROOT_dobj_det+ROOT_nsubj_det			✓						✓	✓	✓	
ROOT_aux+ROOT_dobj_det+ROOT_punct			✓			✓						
ROOT_aux+ROOT_punct+ROOT_punct			✓				✓		✓	✓		✓
ROOT_auxpass			✓						✓			✓
ROOT_ccomp_aux			✓			✓	✓	✓	✓		✓	
ROOT_dobj_acl_aux			✓									
ROOT_dobj_acl_aux+ROOT_dobj_det			✓				✓		✓			
ROOT_dobj_acl_aux+ROOT_dobj_det+ROOT_punct			✓						✓			
ROOT_dobj_acl_aux+ROOT_nsubj_det			✓			✓						
ROOT_dobj_acl_aux+ROOT_punct			✓						✓	✓		
ROOT_dobj_acl_dobj_det			✓			✓			✓			✓
ROOT_dobj_det		✓	✓						✓			
ROOT_dobj_det+ROOT_nsubj_det			✓			✓			✓			
ROOT_dobj_det+ROOT_nsubj_det+ROOT_punct			✓			✓				✓		✓

Table B. Functional Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
ROOT_dobj_det+ROOT_punct			✓									
ROOT_nsubj			✓				✓			✓	✓	✓
ROOT_nsubj_det		✓								✓		✓
ROOT_nsubj_det+ROOT_punct+ROOT_punct			✓			✓	✓		✓	✓		
ROOT_nsubj_nummod			✓			✓	✓		✓	✓		
ROOT_prep_pobj_compound			✓			✓	✓		✓		✓	
ROOT_prep_pobj_det			✓				✓			✓		
ROOT_prep_pobj_det+ROOT_punct			✓						✓			
ROOT_prep_pobj_nummod			✓						✓		✓	
ROOT_punct		✓										
ROOT_punct+ROOT_punct			✓						✓	✓		✓
ROOT+det+dobj			✓							✓		✓
ROOT+dobj			✓						✓			
ROOT+nummod			✓				✓	✓	✓	✓		
ROOT+nummod+punct			✓			✓				✓		
SubTrees		✓	✓			✓			✓			
TreeHeight		✓	✓							✓		

Table B. Functional Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
`Words`		✓	✓			✓	✓					

APPENDIX C: QUALITY FEATURES PER FEATURE SET

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
acl	✓	✓	✓	✓	✓							
AComp		✓	✓	✓	✓			✓	✓	✓		
Adjective		✓	✓						✓			✓
advcl	✓		✓						✓			
Adverb		✓	✓	✓	✓		✓		✓	✓		✓
advmod	✓		✓						✓			
AdvMod	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
amod	✓		✓						✓			
AMod	✓	✓	✓	✓	✓			✓	✓			
aux	✓	✓										
aux+det+dobj			✓									
aux+dobj			✓						✓			
aux+nummod			✓						✓			
aux+nummod+punct			✓						✓			
aux+ROOT+nummod			✓						✓			
auxpass	✓		✓	✓	✓					✓		

Table C. Quality Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
Cardinal		✓	✓	✓	✓	✓	✓			✓	✓	
CompSupAdj		✓	✓						✓			✓
CompSupAdv		✓	✓						✓			
det	✓	✓										
det+aux+nummod			✓				✓		✓		✓	
det+nummod			✓					✓	✓			
det+nummod+punct			✓					✓				
det+ROOT+nummod			✓					✓	✓	✓		
dobj	✓	✓	✓	✓	✓			✓	✓			
dobj+pobj			✓						✓			
DTreeHeight		✓	✓					✓	✓			
hasFverb					✓							
hasQverb					✓							
Length		✓	✓			✓	✓	✓		✓	✓	✓
Modal		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
nmod	✓		✓	✓	✓				✓			
Noun		✓	✓						✓			

Table C. Quality Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
nsubj	✓	✓	✓	✓	✓			✓				
nsubj+aux+dobj			✓				✓					
nsubj+det+dobj			✓					✓	✓			
nsubj+dobj			✓			✓			✓		✓	
nsubj+nummod			✓			✓			✓			
nsubjpass	✓		✓	✓	✓			✓				
nummod	✓		✓	✓	✓							
nummod+pobj			✓					✓	✓			
nummod+punct			✓									
pobj	✓	✓		✓	✓							
prep	✓	✓		✓	✓							
punct	✓	✓										
ROOT_aux		✓										
ROOT_aux+ROOT_aux			✓									
ROOT_aux+ROOT_aux+ROOT_nsubj_det			✓				✓			✓		
ROOT_aux+ROOT_aux+ROOT_punct			✓						✓			
ROOT_aux+ROOT_dobj_acl_aux			✓						✓	✓		

Table C. Quality Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
ROOT_aux+ROOT_dobj_acl_aux+ROOT_dobj_det			✓				✓					
ROOT_aux+ROOT_dobj_acl_aux+ROOT_punct			✓									
ROOT_aux+ROOT_dobj_det			✓				✓		✓			
ROOT_aux+ROOT_dobj_det+ROOT_nsubj_det			✓									
ROOT_aux+ROOT_dobj_det+ROOT_punct			✓									
ROOT_aux+ROOT_punct+ROOT_punct			✓				✓		✓			
ROOT_auxpass			✓				✓					
ROOT_ccomp_aux			✓			✓	✓	✓	✓	✓	✓	✓
ROOT_dobj_acl_aux			✓									
ROOT_dobj_acl_aux+ROOT_dobj_det			✓									
ROOT_dobj_acl_aux+ROOT_dobj_det+ROOT_punct			✓						✓			✓
ROOT_dobj_acl_aux+ROOT_nsubj_det			✓						✓			
ROOT_dobj_acl_aux+ROOT_punct			✓									
ROOT_dobj_acl_dobj_det			✓									
ROOT_dobj_det		✓	✓							✓		
ROOT_dobj_det+ROOT_nsubj_det			✓					✓				✓
ROOT_dobj_det+ROOT_nsubj_det+ROOT_punct			✓									

Table C. Quality Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
ROOT_dobj_det+ROOT_punct			✓							✓		
ROOT_nsubj			✓						✓			
ROOT_nsubj_det		✓										
ROOT_nsubj_det+ROOT_punct+ROOT_punct			✓						✓			✓
ROOT_nsubj_nummod			✓						✓			
ROOT_prep_pobj_compound			✓						✓			
ROOT_prep_pobj_det			✓									
ROOT_prep_pobj_det+ROOT_punct			✓				✓					
ROOT_prep_pobj_nummod			✓						✓			
ROOT_punct		✓										
ROOT_punct+ROOT_punct			✓									
ROOT+det+dobj			✓			✓						✓
ROOT+dobj			✓					✓	✓	✓		
ROOT+nummod			✓									✓
ROOT+nummod+punct			✓									
SubTrees		✓	✓			✓		✓	✓	✓		
TreeHeight		✓	✓							✓		

Table C. Quality Features per Feature Set (cont.)

Feature	FS1	FS2	FS3	FS4	FS5	FS6	FS7	FS8	FS9	FS10	FS11	FS12
`Words`		✓	✓			✓			✓			