Requirements Classification with Interpretable Machine Learning and Dependency Parsing

Fabiano Dalpiaz
Utrecht University
Utrecht, The Netherlands
f.dalpiaz@uu.nl

Davide Dell'Anna
Utrecht University
Utrecht, The Netherlands
d.dellanna@uu.nl

Fatma Başak Aydemir
Boğaziçi University
İstanbul, Turkey
basak.aydemir@boun.edu.tr

Sercan Çevikol
Boğaziçi University
İstanbul, Turkey
sercan.cevikol@boun.edu.tr

***Abstract*—Requirements classification is a traditional application of machine learning (ML) to RE that helps handle large requirements datasets. A prime example of an RE classification problem is the distinction between functional and non-functional (quality) requirements. State-of-the-art classifiers build their effectiveness on a large set of word features like text n-grams or POS n-grams, which do not fully capture the essence of a requirement. As a result, it is arduous for human analysts to interpret the classification results by exploring the classifier's inner workings. We propose the use of more general linguistic features, such as dependency types, for the construction of interpretable ML classifiers for RE. Through a feature engineering effort, in which we are assisted by modern introspection tools that reveal the hidden inner workings of ML classifiers, we derive a set of 17 linguistic features. While classifiers that use our proposed features fit the training set slightly worse than those that use high-dimensional feature sets, our approach performs generally better on validation datasets and it is more interpretable.**

***Index Terms*—requirements engineering, interpretable machine learning, requirements classification, explainable AI**

I. Introduction

Classification is a key example of a machine learning (ML) task applied to requirements engineering (RE). Over the past decades, classifiers have been devised and utilized for many purposes, including the detection and categorization of non-functional requirements (NFRs) [1]–[4], the analysis of app reviews to distinguish between bugs, features, and praises [5] or to identify useful reviews [6], and the separation of requirements from information in large specification documents [7].

The many existing requirements classification approaches differ in the features that guide the classifier, the employed algorithm (Support Vector Machines, decision trees, Naïve Bayes, etc.), the performance metrics, and the training and validation datasets [3]. In spite of this diversity, the most effective classifiers [2] rely on a high-dimensional word feature set: a large number of features are used to guide the classifier (100 to 500 in [2]), and these features are at the word level, e.g., text n-grams or Part-of-Speech (POS) n-grams.

The *many features* hinder the interpretability of those classifiers, making it hard for analysts to understand *why* the classifier performs well or poorly, for the algorithm's decision rules may combine hundreds of features. Furthermore, the *low-level features* characterize requirements at the word level, rather than focusing on the meaning of sentences. We propose the use of interpretable ML [8], [9] to obtain a deeper understanding of the implicit classification rules of automated classifiers.

We choose to focus on the distinction between functional and non-functional requirements, a widely studied topic [3] whose theoretical underpinnings—the distinction between functional requirements, NFRs, quality goals, quality constraints, and functional constraints—are still being discussed [10]–[12]. We envision that interpretable classifiers can be used to shed additional light on this scientific discussion.

In this paper, we lay the foundations for constructing interpretable classifiers that can be used for classification problems in RE. We make the following contributions:

- We manually annotate 1,500+ requirements that consist of the widely used PROMISE dataset [13] as well as requirements from seven other industrial projects. Our annotations use Li *et al.*'s taxonomy [11], which allows a requirement to possess both functional and quality aspects.
- To define our baseline, we reconstruct a word-level, high-dimensional classification approach [2], make it publicly available, and apply it to our annotated dataset. The results confirm its good performance on the training set, and show limited generality when applied to other datasets.
- We propose a new type of classification approach for RE problems that relies on a lower number of more general, thus interpretable, linguistic features such as dependency types [14]. We select our features by using interpretable ML techniques [9] that provide if-then-else rules and helped us introspect how the features guide a classifier.
- We compare our new feature set against our baseline. Quantitatively, the performance is generally comparable; qualitatively, instead, classifiers that use our feature set can be more easily interpreted.

For RE practitioners, we empirically show that it is possible to build more general classifiers that rely on high-level linguistic features without compromising accuracy. These classifiers exhibit a more graceful performance degradation when applied to new datasets, and their inner working are easier to inspect.

Organization. Sec. II sets our theoretical framework for tagging functional and quality aspects. Sec. III defines our baseline, by assessing the performance of a high-dimensional feature set on seven industrial datasets. Sec. IV describes the creation of our proposed feature set. Sec. V analyzes our feature set and discusses the impacts on RE. Sec. VI reviews the main threats to validity, Sec. VII reviews related work, while Sec. VIII presents conclusions and future directions.

2332-6441/19/$31.00 ©2019 IEEE
DOI 10.1109/RE.2019.00025

II. The F/NFR classification problem, revisited

Building on the recent literature on functional and quality requirements, we detail our version of the classification problem (Sec. II-A), which we apply to eight datasets (Sec. II-B). Finally, we report on the tagging process of the datasets (Sec. II-C).

A. Building the classification problem

Li *et al.* [11] adopt a quality-oriented approach to model NFRs as quality goals. A quality is a basic perceivable or measurable characteristic that inheres in and existentially depends on its subject [15], 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 from 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 [7].

B. Datasets

We obtained eight datasets and manually classified them using the approach described in Sec. II-A. PROMISE [13] collects 625 requirements from 15 projects, created and classified by students, and has previously been used to train and test other requirements classification approaches (e.g., [2], [11]). The ESA Euclid dataset was provided to us by the European Space Agency and concerns the Euclid mission [16]; we have selected 236 requirements from the system requirements document, retaining those in the sections: reliability and safety, safe mode, altitude and orbit control, propulsion, telemetry, tracking and command. We also use two other private datasets from the IT domain, which include 172 and 138 requirements, respectively. The first dataset contains the user requirements for implementing an off-the-shelf Helpdesk system, and the second dataset has requirements for a bespoke user account request and management application (User mgmt). The Dronology dataset [17] has 97 system requirements for Unmanned Aerial Systems (UASs). The ReqView dataset [18] details the requirements specification for the ReqView requirements management tool. for ReqView, we have converted the format of the requirements by combining the modal verbs column (‘shall’, ‘could’) with the requirement text (‘print a report’, ‘be usable’) into full sentences (‘the system shall print a report’). The requirements for Leeds University’s Library online management system (Leeds library) are documented in an online spreadsheet [19]. We have removed the lines of text that were not related to requirements. The final dataset consists of requirements for the Web Architectures for Services Platforms (WASP) application [20].

C. Tagging

Each dataset was tagged by two authors independently. Then, the taggers organized reconciliation meetings to go over the disputes in tagging. If the taggers failed to convince each other, a third author was consulted for the final tag. The authors went over all disputes and managed to resolve them.

Table I
Krippendorff’s α and agreement percentage for the eight datasets

Dataset	F		Q		Dataset	F		Q	
	Kα	%	Kα	%		Kα	%	Kα	%
PROMISE	-0.38	0.45	0.67	0.84	Dronology	-0.12	0.77	0.41	0.76
ESA Euclid	0.68	0.84	0.43	0.87	ReqView	0.64	0.91	0.74	0.89
Helpdesk	0.67	0.90	0.85	0.94	Leeds library	0.65	0.82	0.54	0.80
User mgmt	0.71	0.95	0.70	0.92	WASP	0.52	0.90	0.37	0.71

Table I shows the initial level of agreement between the taggers using the Krippendorff’s α (KALPHA) metric [21], which measures the observed disagreement normalized to the expected disagreement. Only one of the 16 scores is above the customary threshold of 0.8 and six in total are greater than the minimum acceptable limit of 0.667. One explanation on the low agreement score for the PROMISE dataset is that our taggers used it as the training dataset. The imbalanced nature of the datasets should not be ignored when interpreting the level of agreement as it disturbs the reliability of the KALPHA metric. One extreme case is the negative result obtained for the Dronology dataset, which has 94 F tags in 98 requirements. While the scores presented in Table I prove the difficulty of the requirements classification problem, the taggers were able to resolve all conflicts in the next round, which indicates the significance of clear standards for the classification task.

Table II summarizes the output of the tagging process after the reconciliation sessions. The datasets are ordered by the number of requirement rows. As described above, 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. The tagged datasets are available in our repository [22].

Table II
Overview of the tagged datasets: our gold standard

Dataset	Public	Rows	F	Q	OnlyF	OnlyQ	F+Q	¬R
PROMISE	Yes	625	310	382	230	302	80	13
ESA Euclid	No	236	91	211	23	143	68	2
Helpdesk	No	172	143	51	121	29	22	0
User mgmt	No	138	126	25	113	12	13	0
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		1,502	938	809	674	545	264	19

III. Applying a high-dimensional, word-level feature set to our classification problem

As a first step of our research on interpretable classifiers for RE, we need to select a high-dimensional, word-level feature set to take as a reference in terms of classification performance and interpretability. Among the existing options [3], we choose a recent approach [2] that shows excellent performance and is described extensively in the original paper.

A. Method

Our study involves *i.* reconstructing the feature set, *ii.* using this set in an automated classifier that adopts the same technique as [2], *iii.* training the classifier on a dataset (PROMISE NFR [13], often used in other studies), and *iv.* assessing the performance on a heterogeneous set of requirements datasets. We evaluate the approach in different settings:

Fitness on PROMISE (*train*). How well does the high-dimensional classifier[1] fit the training dataset that is used to construct the classification model?

75%-25% splitting of PROMISE (*test*). A popular way to perform an inexpensive validation by randomly splitting the dataset into two: 75% of the entries are used to train the classifier, the remaining 25% for testing it.

k-fold cross-validation of PROMISE (*kfold*). The dataset 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 remaining k^{th} fold. We employ stratified k-fold, which ensures a similar class ratio (positive/negative) in each of the folds.

Project-level cross-validation (*pfold*). Since PROMISE consists of 15 projects, we use 12 of them as training set, and 3 as a 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 ensured that two projects co-occur in at most one test set.

Industrial datasets (*industry*). The PROMISE-trained classifier is evaluated on the seven industrial datasets of Sec. II. We train the classifier on PROMISE to study its adaptability to requirements from different projects.

B. Metrics and tools

While Kurtanović and Maalej [2] had a single binary classification problem (F vs. NFR) and used metrics for each of the two classes F and NFR, our framework of Sec. II 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 only functional aspects?
OnlyQ: does a requirement possess only quality aspects?

For each of these settings, we employ a combination of metrics that are widely used in ML to assess the performance of binary classifiers. Besides the metrics of precision, recall, and F_1 score[2], which are commonly used in the RE literature, we use the receiver operating characteristic (ROC) plot and its associated metric, the area under the ROC curve (AUC).

ROC plots [24] are 2-dimensional charts that show the trade-off between recall (y-axis) and specificity (the true negatives rate, x-axis). The so-called 'ROC heaven' is the top-left corner, in which recall is 1.0 and there are no false positives, i.e., precision is also 1.0. In ROC plots, classifiers are represented as a line that is plotted by calculating recall and 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 [25], 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. Fig. 1 illustrates these notions.

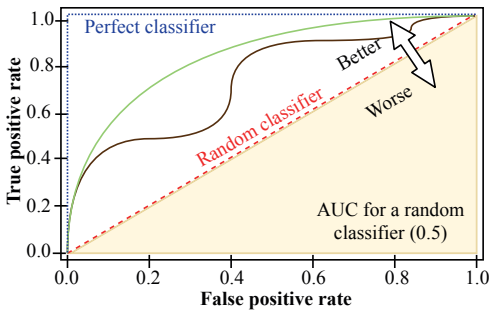

Figure 1. Illustration of the ROC plot and of the AUC

The ROC curve has some limitations with unbalanced datasets [26]. Thus, we complement it with an analysis of precision for unbalanced datasets (e.g., in Table II, Dronology has only 2 requirements tagged as OnlyQ). We resort to oversampling to plot smoother ROC curves when the ratio between the two classes exceeds 10:1 and the minority class has 6+ requirements. These values are recommended guidelines for ADASYN [27], a state-of-the-art oversampler that generates synthetic samples by taking the k closest neighbors as input.

C. Reconstructing Kurtanović and Maalej's approach

We could not access the original classifier by Kurtanović and Maalej [2], for it is not available online and the authors could not give us access to a working copy. Fortunately, the original publication [2] is relatively clear on the feature set, and we complemented this knowledge with the classifier that

[1]For brevity, we use the term 'high-dimensional classifier' to indicate a 'classifier trained on a high-dimensional feature set'.

[2]We do not study the choice of an appropriate β for the F_β metric [23], for its tuning would require us to study the impact of precision and recall on the daily practices of the development teams who make use of the datasets.

Table III
PERFORMANCE OF OUR REPRODUCTION OF [2] FOR THE FOUR CLASSIFIER TARGETS: F, Q, ONLYF, ONLYQ

(a) Top 500 features

Test set	F				Q				OnlyF				OnlyQ			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
PROMISE train	0.981	0.984	0.982	1.00	0.985	1.000	0.990	1.00	1.000	0.987	0.995	1.00	0.990	0.964	0.978	1.00
PROMISE test	0.819	0.797	0.822	0.89	0.909	0.891	0.873	0.92	0.870	0.870	0.911	0.94	0.896	0.852	0.873	0.91
PROMISE k-fold	0.755	0.684	0.712	0.80	0.785	0.867	0.822	0.84	0.766	0.630	0.681	0.86	0.741	0.798	0.766	0.82
PROMISE p-fold	0.749	0.602	0.663	0.78	0.714	0.877	0.781	0.80	0.752	0.475	0.573	0.81	0.683	0.794	0.728	0.81
Industry	0.773	0.716	0.693	0.69	0.483	0.624	0.550	0.58	0.607	0.414	0.575	0.60	0.415	0.497	0.706	0.69
Industry (std-dev)	±.18	±.15	±.09	±.06	±.23	±.08	±.12	±.07	±.23	±.12	±.09	±.05	±.28	±.20	±.08	±.09
ESA Euclid	0.477	0.451	0.597	0.59	0.898	0.706	0.665	0.54	0.068	0.174	0.686	0.48	0.708	0.524	0.581	0.64
Helpdesk	0.903	0.972	0.890	0.78	0.542	0.510	0.727	0.67	0.785	0.843	0.727	0.69	0.368	0.241	0.802	0.72
User mgmt	0.583	0.643	0.594	0.65	0.151	0.560	0.348	0.47	0.872	0.301	0.391	0.63	0.622	0.578	0.610	0.61
Dronology	1.000	0.670	0.680	0.77	0.370	0.607	0.588	0.66	0.783	0.529	0.567	0.66	0.050	0.500	0.794	0.74
ReqView	0.898	0.707	0.678	0.66	0.409	0.562	0.540	0.58	0.639	0.426	0.494	0.60	0.333	0.636	0.793	0.70
Leeds library	0.654	0.773	0.671	0.72	0.700	0.689	0.565	0.52	0.250	0.217	0.612	0.51	0.645	0.500	0.635	0.73
WASP	0.898	0.800	0.742	0.66	0.311	0.737	0.419	0.61	0.850	0.405	0.548	0.64	0.176	0.500	0.726	0.66

(b) Top 100 features

Test set	F				Q				OnlyF				OnlyQ			
	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC	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	0.927	0.887	0.933	0.98	0.884	0.884	0.888	0.95
PROMISE test	0.795	0.784	0.803	0.90	0.910	0.901	0.879	0.95	0.863	0.815	0.892	0.96	0.859	0.753	0.809	0.90
PROMISE k-fold	0.819	0.742	0.774	0.87	0.817	0.909	0.858	0.89	0.818	0.674	0.732	0.90	0.785	0.808	0.795	0.82
PROMISE p-fold	0.805	0.699	0.742	0.85	0.752	0.917	0.823	0.85	0.816	0.515	0.616	0.86	0.745	0.802	0.770	0.81
Industry	0.851	0.714	0.740	0.80	0.533	0.568	0.591	0.61	0.618	0.610	0.651	0.63	0.527	0.472	0.777	0.78
Industry (std-dev)	±.16	±.13	±.08	±.11	±.23	±.09	±.12	±.09	±.32	±.25	±.11	±.10	±.30	±.26	±.10	±.08

the same authors developed to classify app reviews [28], whose code is partially available online [29].

We applied a few minor modifications to the original version:

- Parse trees were built using Berkley's *benepar* library, a state-of-the-art constituency parser that outperforms [30] the Stanford parser used in [2].
- 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 dataset taken from Amazon software reviews that the authors had used to artifically balance the minority class of NFRs.
- 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 is fully available in our online repository [22] as two Jupyter Notebooks.

D. Results

We analyze the results from our experiments: precision, recall, and F1 score are reported in Table III for two high-dimensional feature sets that include the most informative features automatically selected through scikit-learn libraries: the top-500 and the top-100. As a classifier, we use scikit-learn's Support Vector Machines (SVM) implementation that we executed with the linear kernel. This is the same classifier that was used in [2]. Due to space limitations, we show only summary data for top-100. The ROC plot for top-500 is shown in Fig. 2. The remaining data are available online.

A first set of observations can be drawn from the analysis of precision, recall, and F1 score:

Q vs. F performance (top-500). For the top-500 features setting (Table III.a), the results for Q outperform those for F and OnlyF on the PROMISE-derived validation datasets (test, k-fold, and p-fold), especially in terms of recall. Conversely, in the industrial datasets, the results for Q are way worse.

F: the easiest target? In both top-100 and top-500, the classifiers perform best on the F class, especially in industry datasets. This probably occurs because F is the majority class: 938/1502 requirements possess a functional aspect.

Quality aspects beyond training data. The performance of the classifiers in identifying quality aspects (Q, OnlyQ) degrades considerably with the industry datasets. One key example is Q, losing $\sim$0.2–0.3 in recall and precision both with top-500 and top-100, compared to PROMISE. Also, the recall for OnlyQ worsens $\sim$0.3 with both top-100 and top-500.

The difficult targets: Only-*. The performance for OnlyF and OnlyQ on industrial datasets shows high variance; the standard deviation for their precision is in the $[0.23, 0.32]$ range. An indication that the feature set did not lead to a general classifier.

Additional observations on the robustness of the classifiers can be made through the ROC/AUC analysis of Fig. 2:

The difficult ESA Euclid case. The ROC plots show a poor curve for ESA Euclid, with an AUC score between 0.48 and 0.64, not much better than a classifier with a random outcome (0.5). The results need some interpretation. For ESA Euclid, the class Q is dominant over OnlyF (211 vs. 23), and we cannot therefore draw strong conclusions. Nonetheless, the results for

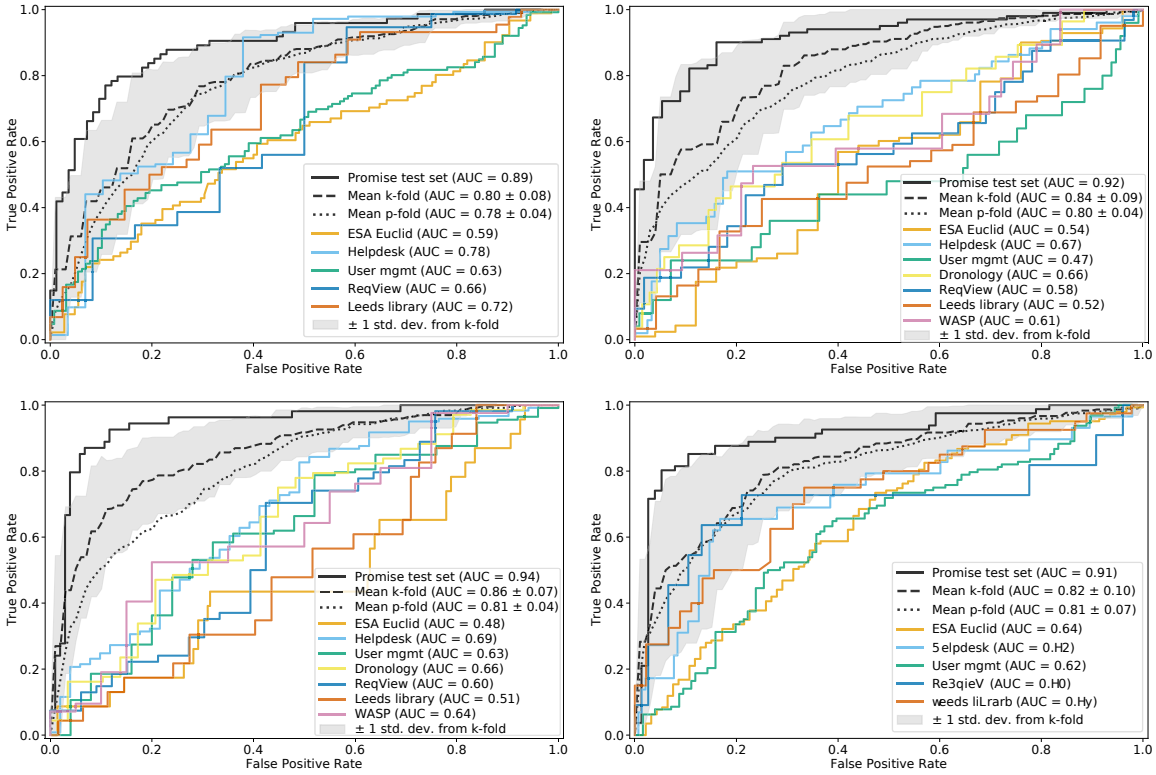

Figure 2. ROC plots for the top 500 features: F (top-left), Q (top-right), OnlyF (bottom-left), OnlyQ (bottom-right)

the more balanced F and OnlyQ classes are also disappointing.

Low generality for Q and OnlyF. The ROC plots that represent Q (top-right) and OnlyF (bottom-left) highlight a clear degradation when applying the classifier, trained on PROMISE, to the industrial datasets. See how the curves for the industrial datasets fall outside the standard deviation of k-fold.

IV. Towards More Informative Features for Automated Classification of Requirements

To cope with the mixed results on the industrial datasets, we propose the use of higher-level linguistic features in the feature set. After introducing dependency parsing and describing initial experiments (Sec. IV-A), we use interpretable ML tools to select a feature subset that performs well and has low size (Sec. IV-B), and a further attempt to improve the performance with quality aspects (Sec. IV-C).

A. Dependency Parsing and Linguistic Features

Dependency parsing [14] is the task of identifying the grammatical structure of a sentence by determining the linguistic dependencies between the words, according to a pre-defined set of *dependency types*. For instance, in the requirement 'The system shall refresh the display every 60 seconds', 'display' is the direct object (*dobj*) of the main (root) verb 'refresh', 'shall' is the auxiliary verb (*aux*) that affects 'refresh', '60' is the numeric modifier (*nummod*) of 'seconds', etc.

We hypothesize that the presence of linguistic features like dependency types can help determine whether a requirement contains functional or quality aspects. We describe the feature engineering process we carried out to identify features based on dependency types, and we present three selected feature sets that we tested on PROMISE and on the industrial datasets through an SVM classifier configured as in Sec. III-C. We use the dependency parsing implementation of the spaCy toolkit.

We consider two categories of features from dependency parsing: *i. dependency types* themselves (e.g., *nummod* and *dobj*); and *ii. branch types of the dependency tree* of a requirement. For example, in the sentence 'refresh the display', a branch of type *ROOT* $\rightarrow$ *dobj* $\rightarrow$ *det* connects the root verb 'refresh' to the determinant 'the' via the direct object 'display'.

We analyzed the eight datasets to determine the linguistic features that appeared more often and in which type of requirements (F, Q, F+Q, OnlyF or OnlyQ). For each type of requirement, we computed the coverage of each feature, i.e., the ratio between the number of requirements that possess that feature over the number of requirements. For example, 90% of all requirements contain the *dobj* dependency type.

We also analyzed the percentage of requirements containing combinations of two or three features together. For instance, the combination of *dobj* and *det* appears in 60% of OnlyF requirements but only in 30% OnlyQ requirements.

In total, we analyzed six groups of linguistic features: *(1)* single dependency types; combinations of *(2)* two and *(3)* three

dependency types; *(4)* types of branches; combinations of *(5)* two and *(6)* three types of branches together.

To determine which features could help a classifier distinguish between functional and quality aspects, for each of the six groups of linguistic features, we calculated the coverage difference between OnlyQ and OnlyF as follows: $\Delta = |cov(OnlyQ) - cov(OnlyF)|$. The linguistic features with the highest Δ values are those features that are typically present in OnlyQ but not in OnlyF and *vice versa*.

We inspected the 10 most significant features (highest Δ) for each of the six groups and we obtained a set of 60 different linguistic features, shared in our public repository [22]. We tested their performance with various feature subsets, and this led to three feature sets that performed better on average:

FS1. The top 10 single dependency features of group *(1)* plus the single dependency features that were not in the top 10 of *(1)* but appeared at least once in the top 10 features of groups *(2)* and *(3)*. This set includes 17 features.

FS2. We take the top 10 features of each of the six groups, and we retain only those features with $\Delta > 0.2$, obtaining a total of 12 features. We then add other features that have been used in the literature [2], [31]: *i.* length of the requirement in characters; the number of *ii.* modals, *iii.* adjectives, *iv.* nouns, *v.* adverbs, *vi.* cardinals, *vii.* comparative and superlative adjectives, *viii.* comparative and superlative adverbs, *ix.* words; *x.* 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. Finally, we added three dependency type-based features that we identified through a qualitative exploration of the dataset as promising indicators of quality aspects:

xiii. Nr. of adverbial modifiers that link a verb to an adverb:

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

xiv. Nr. 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. Nr. 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, FS2 includes $12 + 12 + 3 = 27$ features.

FS3. The top 10 features of each of the six groups, plus the features *i-xv*. In total, FS3 has $60 + 12 + 3 = 75$ features.

Table IV reports the results obtained using feature sets FS1, FS2, and FS3, and includes the outputs for PROMISE test and the macro-average of the results for the industry datasets. A comparison between Table III and Table IV reveals a similar performance, despite the considerably smaller size of our feature sets. The feature reduction is made possible by the higher abstraction level of dependency types compared to lexical features. A single dependency type subsumes multiple lower-level text n-grams or POS n-grams. For example, a fragment like 'print a report' leads to the dependency parsing tree $dobj \rightarrow det$, but the same fragment would be captured as twelve word-level features in [2]: six text n-grams: 'print', 'a', 'report', ('print', 'a'), ('a', 'report'), ('print', 'a', 'report'); and six POS n-grams: VERB, DET, NOUN, (VERB, DET), (DET, NOUN), and (VERB, DET, NOUN). Also, a variant like 'save the page' would introduce six additional text n-gram features.

B. Feature selection via interpretable ML

The black-box nature of ML classifiers like SVM or neural networks makes it difficult for humans to comprehend the logic behind the predictions. With the growing adoption of ML techniques, there is an increasing trend towards making ML systems more transparent and interpretable [32].

We employ two recent tools that facilitate the interpretation of ML classifiers. *RuleMatrix* [9] extracts logic rules that approximate a black-box classifier using its feature set. RuleMatrix also shows the rules on an interactive matrix-based visualization that depicts which rules apply to the data, their support, fidelity to the original classifier, and accuracy. Such visualization aids human analysts understand, explore, and validate predictive models. *SkopeRules* [33] 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, and the best rules are selected based on their performance trying to avoid duplicate rules.

We leverage the interpretability capabilities of SkopeRules and RuleMatrix to analyze the predictions made by employing the three feature sets FS1, FS2, FS3. Three authors of the paper ran the tools, analyzed the generated logic rules, and identified those features that appeared commonly and that helped distinguish functional and quality aspects.

Below, we show a set of rules produced by SkopeRules using FS1 to classify functional aspects in the PROMISE dataset. If a requirement satisfies any of the rules, it is classified as F.

1. $\neg advmod \wedge dobj \wedge nsubj \wedge \neg nsubjpass \wedge \neg nummod$
2. $\neg acl \wedge dobj \wedge \neg nmod \wedge nsubj \wedge \neg nummod$
3. $acl \wedge dobj \wedge nsubj \wedge \neg nummod \wedge pobj$

The first rule states that a requirement has functional aspects when: an adverbial modifier (*advmod*), a passive nominal subject (*nsubjpass*), and a numeric modifier (*nummod*) are not present, but a direct object dependency (*dobj*) and an active nominal subject (*nsubj*) exist. A requirement that satisfies this rule is "The WASP platform must have a web-based form to specify simple POI and service profiles": the dependencies are shown in Fig. 5 later in this paper. With these three rules that refer to just eight features, SkopeRules reaches a precision of 0.69 and a recall of 0.73 on the PROMISE test set.

The following rules are generated by RuleMatrix with FS1 for the F task on PROMISE. Each 'if' statement denotes one rule. Rules are therefore applied in cascade: a rule can fire only if none of the previous ones fired. RuleMatrix associates with each rule a pair `[p,q]`: the probability `q` for the target class, and probability `p` for non-target class.

Table IV
PRECISION, RECALL, F1-SCORE, AND AUC ON PROMISE AND INDUSTRY MACRO-AVG WITH THE THREE FEATURE SETS FS1, FS2, FS3

Target	Test set	FS1 (17 features)				FS2 (27 features)				FS3 (75 features)			
		Prec	Rec	F1	AUC	Prec	Rec	F1	AUC	Prec	Rec	F1	AUC
F	PROMISE test	0.67	0.74	0.71	0.73	0.67	0.81	0.72	0.76	0.68	0.78	0.73	0.77
	Industry	0.81 ±.16	0.87 ±.08	0.79 ±.09	0.72 ±.09	0.78 ±.17	0.93 ±.08	0.80 ±.13	0.74 ±.09	0.80 ±.18	0.90 ±.10	0.80 ±.12	0.74 ±.11
Q	PROMISE test	0.87	0.73	0.76	0.78	0.85	0.78	0.77	0.81	0.88	0.81	0.81	0.85
	Industry	0.63 ±.20	0.29 ±.11	0.64 ±.12	0.60 ±.06	0.58 ±.24	0.28 ±.15	0.62 ±.13	0.60 ±.08	0.61 ±.24	0.40 ±.12	0.65 ±.07	0.61 ±.06
OnlyF	PROMISE test	0.59	0.78	0.74	0.79	0.65	0.78	0.78	0.83	0.69	0.78	0.80	0.86
	Industry	0.83 ±.16	0.87 ±.08	0.79 ±.09	0.68 ±.12	0.83 ±.18	0.82 ±.12	0.77 ±.10	0.77 ±.07	0.82 ±.20	0.77 ±.14	0.73 ±.09	0.68 ±.10
OnlyQ	PROMISE test	0.73	0.68	0.71	0.74	0.78	0.62	0.71	0.76	0.76	0.65	0.71	0.77
	Industry	0.68 ±.20	0.29 ±.12	0.63 ±.10	0.58 ±.06	0.76 ±.20	0.17 ±.10	0.59 ±.16	0.57 ±.11	0.73 ±.23	0.23 ±.11	0.62 ±.10	0.60 ±.06

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

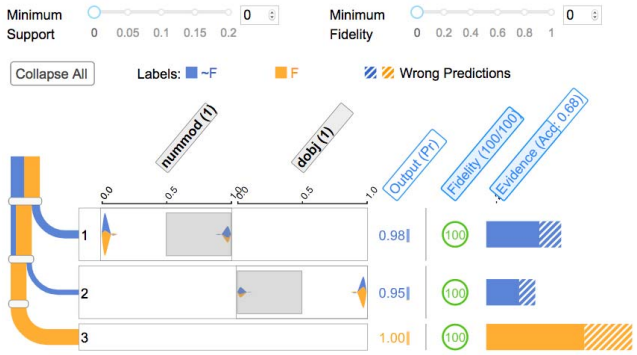

Figure 3. Interactive rule visualization interface of RuleMatrix

The effectiveness of the rules can be introspected through RuleMatrix's interactive visualization: see Fig. 3. Each row represent one rule. The waterflow diagram on the left represents all the data, and the horizontal flow to the rule captures how many data are classified by that rule. Each column corresponds to a feature (here, *nummod* and *dobj*). The intersection of a rule and a feature is marked with a gray rectangle if the rule refers to that feature. Hovering on the area reveals the rule that relates to that feature. The `Output` column shows the probability that the data belongs to the target class (orange) or non-target class (blue), according to the rule. The `Fidelity` column indicates the degree to which the rule aligns with the outcome from the original classifier. Finally, the column `Evidence` describes how much data are correctly and wrongly predicted.

The three rules identified by RuleMatrix, for example, are similar to, but simpler than those produced by SkopeRules. RuleMatrix's rule set uses only two features: *dobj* and *nummod*, and the rules reach a performance that is very similar to FS1's classifier on PROMISE test for the F problem: a precision of 69% and a recall of 73% with just 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.

Through a qualitative analysis of the logical rules generated by SkopeRules and RuleMatrix, we selected 15 significant features. We then used this feature set to classify the requirements. Table V reports the results, and shows, again, comparable results to those obtained with the feature sets FS1, FS2, FS3 and shown in Table IV, with some oscillations such as -0.06 for the precision of Q, $+0.07$ for the recall of Q, and -0.09 for the recall of OnlyF in industry.

In comparison to the high-dimensional classifiers of Table III, the performance with F and OnlyF is better, on average, on industrial datasets. In particular, there is a recall gain of $+0.2$ in F, and a precision gain of $+0.2$ and a recall gain of $+0.4$ in OnlyF. On the PROMISE dataset, our performance is generally lower in the range of -0.05 to -0.15. The major degradation concerns Q and OnlyQ, in which we consistently score lower in terms of recall (between -0.1 and -0.3) both on PROMISE and on the industry datasets. On the other hand, major degradations in recall (-0.3 with the industry datasets for OnlyQ) are compensated by precision gains ($+0.25$).

Table V
PREC, REC, F1 AND AUC WITH THE 15 LINGUISTIC FEATURES IDENTIFIED WITH SKOPERULES AND RULEMATRIX

Target	Test set	Prec	Rec	F1	AUC
F	PROMISE test	0.67	0.74	0.71	0.76
	Industry	0.80 ±.16	0.87 ±.08	0.79 ±.09	0.70 ±.14
Q	PROMISE test	0.87	0.76	0.77	0.80
	Industry	0.57 ±.18	0.36 ±.14	0.65 ±.09	0.62 ±.06
OnlyF	PROMISE test	0.63	0.80	0.77	0.83
	Industry	0.82 ±.19	0.78 ±.15	0.75 ±.10	0.69 ±.13
OnlyQ	PROMISE test	0.74	0.69	0.71	0.75
	Industry	0.68 ±.19	0.29 ±.12	0.62 ±.10	0.59 ±.04

C. Improving the identification of quality aspects

The loss of performance with Q and OnlyQ prompted us to conduct an additional attempt. Inspired by earlier work [1] that used keywords as features for NFRs, we combined this idea with the dependency type *root*, the main verb of a sentence.

We searched for root verbs in the datasets that occur significantly more frequently (as a heuristic, 3 times more often) in requirements tagged with quality aspects rather than in requirements tagged with functional aspects, and *vice versa*. We identified a list of verbs with *functional prevalence*, each occurring 3 times more often in OnlyF than in OnlyQ and 3 times more often in F than in Q; and an analogous list of verbs

with *quality prevalence*. We set a minimum threshold of 10 occurrences in our dataset to exclude infrequent verbs.

Based on these verbs, listed in Table VII, we created the boolean features *fverb* and *qverb*, which are true when a requirement's root is a verb in the corresponding list. We then added these two features to the 15 features from the previous section, and we obtained the results shown in Table VI.

Table VI
PREC, REC, F1 AND AUC WITH 17 FEATURES, INCLUDING ROOT VERBS

Target	Test set	Prec	Rec	F1	AUC
F	PROMISE test	0.71	0.76	0.73	0.78
	Industry	0.82 ±.16	0.87 ±.09	0.80 ±.09	0.80 ±.09
Q	PROMISE test	0.77	0.80	0.92	0.80
	Industry	0.55 ±.22	0.70 ±.11	0.65 ±.09	0.68 ±.07
OnlyF	PROMISE test	0.77	0.83	0.72	0.82
	Industry	0.89 ±.16	0.50 ±.17	0.61 ±.11	0.80 ±.08
OnlyQ	PROMISE test	0.71	0.75	0.78	0.64
	Industry	0.73 ±.16	0.32 ±.09	0.64 ±.11	0.66 ±.07

While the results for F are comparable to those of Table V, the additional features provide a slight improvement for OnlyQ and significantly better results for Q: the recall on the industrial datasets 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 Sec. III-C, we observe that:

- On PROMISE, precision and recall worsen, but the degradation is limited (circa −0.1);
- On the industry datasets, our approach shows substantial improvements in recall for F (+0.16) and in precision for OnlyQ (+0.31) and OnlyF (+0.28).
- For the OnlyQ target, our approach worsens in recall (−0.18), but shows a large gain in precision (+0.31).
- The ROC plot of Fig. 4 shows that, for the F case, a classifier with our features trained on PROMISE does not degrade on the industrial datasets. The plots for the other targets show some degradation, like those in Fig. 2.

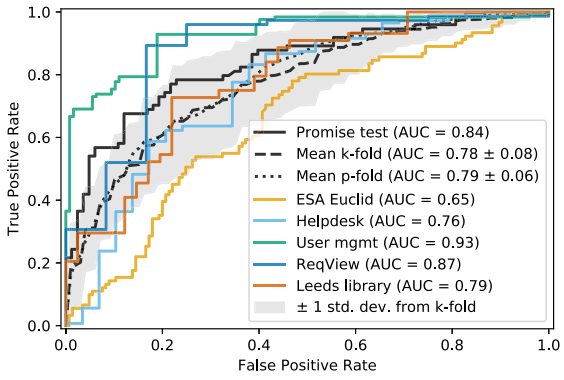

Figure 4. ROC plot for F with the final 17 features

V. FEATURE SET ANALYSIS AND IMPACT ON RE

We first analyze our final feature set and illustrate it on some requirements (Sec. V-A), then compare it with the (POS) n-grams approach (Sec. V-B). Finally, we discuss the impact of our work on RE research and practice in Sec. V-C.

A. Dissecting our Feature Set

Table VII lists the final feature set using definitions adapted from the Universal Dependencies project [34], a world-wide attempt to reconcile the existing dependency parsing tag sets.

Table VII
THE FINAL FEATURE SET, INCLUDING 17 FEATURES

Name	Tag	Description
Boolean features: dependency types		
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 prepositional modifier.
Numeric features		
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).
Boolean features: root verb types		
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}?

We explain the dependency types via four requirements from our datasets. Fig. 5 illustrates a subset of the linguistic dependencies in those sentences, showing only those dependency types that are in Table VII. For example, in Leeds library, 'system' is the nominal subject (*nsubj*) whereas 'metadata' is the direct object (*dobj*) of the main verb 'support', while the adverbial clause (*advmod*) rooted by 'related' modifies 'metadata'. User mgmt has a passive nominal subject (*nsubjpass*) of a passive verb: 'fields' is the subject of 'set'. *Advmod* and *amod* dependencies are created by adverbs and adjectives, that modify verbs and nouns, such as 'automatically set' in

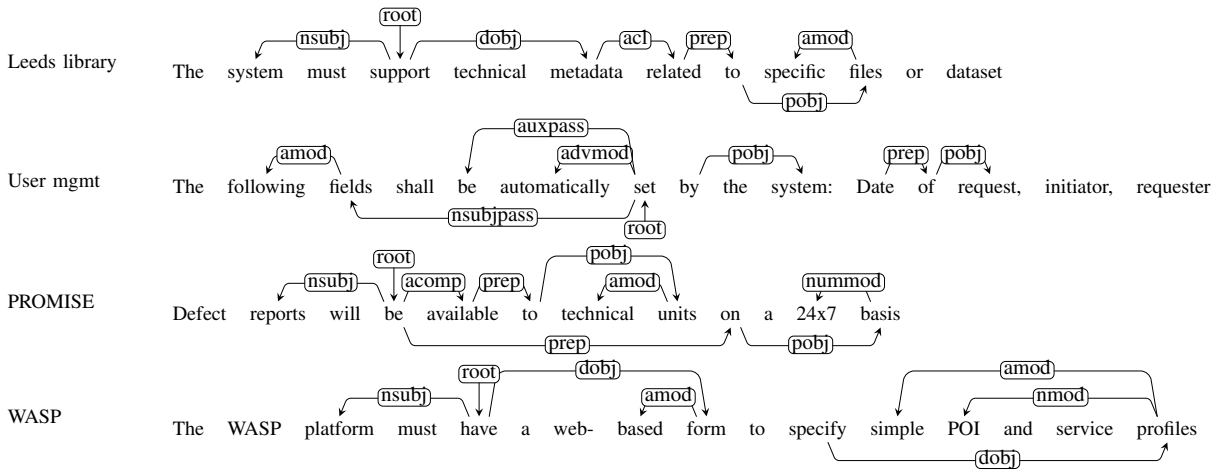

Figure 5. Requirements from our datasets showing the dependency types included as features in our approach

User mgmt and 'technical units' in PROMISE. Nouns can also modify nouns (*nmod*) as in 'POI ... profiles' in WASP.

There are additional four numeric features that count how many instances of *i.* adjectival complements (*acomp*) that affect the root verb (e.g. 'available' in PROMISE), *ii.* cardinals (*CD*) *iii.* modal verbs (*MD*), and *iv.* adverbs (*RB*).

Finally, the features include the verb types that indicate dominance of functional or quality aspects. None of the verbs in *fverb* occur in Fig. 5, while PROMISE's main verb is 'be', indicating a quality trait in the requirement (*qverb*).

The examples show how our feature set provides a rich characterization of each requirement, which includes many relationships between non-contiguous words that cannot be captured easily with an n-gram based approach.

B. Dependency parsing vs. (POS) n-grams

Besides the positive quantitative results (Sec. IV), our approach's added value is that, via interpretable ML techniques that uncover the classification rules, humans can determine which linguistic structures lead to functional or quality aspects. To show this advantage, we ran SkopeRules on the top-100 feature set, and compared the identified rules against our features. We highlight the key findings based on the *statistical* co-occurrence of certain patterns with quality or functional aspects. We use POS tags from the Penn Treebank's POS tagset [35], which was used in both [2] and our reconstruction.

Adverbial modifiers as quality indicators. In the OnlyF top-100 classifier, the POS 3-gram (MD, RB, VB) appears in SkopeRules: when not present, a requirement possesses also quality aspects in 32/37 cases. This 3-gram is a lower-level representation of the *advmod* dependency from VB to RB. However, other n-grams may refer to *advmod*; e.g., the bi-gram (VB, RB) denotes a quality in 41/53 occurrences.

Detecting functional aspects via direct objects. The SkopeRules output for F and OnlyF refer to the n-gram (VB, DT, NN) as a feature that helps identify functional aspects, when present (in 383/473 of the cases). In our feature set, this POS 3-gram is generalized by *dobj*, which is implicitly referred to by other n-grams in the top-100 features like (VB, DT), (VB, DT, JJ), and (MD, VB, DT). The last 3-gram is likely to represent typical cases of a *dobj* that could be captured by a 4-gram, i.e., (MD, VB, DT, NN), e.g., 'shall print a paper'. The *dobj* dependency appears as a determinant for quality aspects in the rules shown in Sec. IV-B.

Numerical modifiers as a quality indicator. In SkopeRules with top-100, the bi-grams (CD, IN) and (CD, NNS) appear often as indicators of quality aspects. The latter bi-gram, which indicates quality aspects in 111/122 occurrences, is a lower-level representation of *nummod*. Think of sentence fragments like '5 degrees', '2 seconds', etc. However, the bi-gram would fail to recognize a fragment like '3 consecutive days', which denotes a *nummod* dependency from 'days' to '3'. In our feature set, as visible by the rules in Sec. IV-B, *nummod* is a likely indicator of quality aspects.

The difficulties with passive sentences. The problem of sequentiality is exacerbated when we consider passive voice sentences, which are frequent in requirements; e.g., '*Users shall feel satisfied* using the product'. 3-grams are unable to capture these cases; however, with our knowledge of dependency types, we could read the output of SkopeRules which includes the 3-gram (VB, VBN, IN): its absence seems to indicate a requirement with also functional aspects. Our features *nsubjpass* and *auxpass* characterize passive voice sentences.

C. Implications on RE practice and research

Our feature set can be used by RE practitioners such as product managers and requirements analysis in different ways.

Bootstrapping a classifier with limited data. The quantitative results and the ROC plots (compare Fig. 4 with Fig. 2) show that our feature set degrades more gracefully than high-dimensional classifiers when applied to different datasets. This could be useful for software organizations who start a new project and wish to use their existing requirements to train their classifier.

No-machine-learning classifier. For organizations that are not

willing to invest in machine learning, the if-then-else rules that can be extracted by our pipeline can be used to build a static classifier that, although imperfect, is inexpensive and exhibits good-enough performance on similarly structured requirements.

Reflecting on requirements authoring. The advanced practitioner can inspect why a requirement is classified in a certain way by checking the interpretable ML rules, and reflect on the linguistic practices of authoring good requirements [36].

Researchers and innovative practitioners can use the feature set for constructing new requirements classifiers that rely on higher-level linguistic features. Our binary classifier can be turned into a recommendation tool that provides degrees of membership for the various aspects: if we take F and Q, the probability assigned by the classifier can be shown explicitly instead of using it as a cutoff. For example, a requirement could have 90% likelihood to have functional aspects, and 60% likelihood to have quality aspects. Furthermore, the list of functional and quality verbs can be customized for the domain of use, for the prevalence of qualities is domain specific [12].

VI. Threats to validity

Conclusion validity. The validity of the statistical results is threatened by the existence of unbalanced datasets, in which not all classes are evenly represented. This is a general problem in the RE field: the scarcity of large-scale datasets makes mitigation techniques like under-sampling impractical.

Internal validity. Although our framework based on Q and F alleviates taggers from taking a sharp decision in the F/NFR dichotomy, the results of Table I show the difficulty in obtaining a consistent coding. We mitigated this through the tagging reconciliation meetings, yet it is still plausible that other taggers may have produced a different gold standard, and that the results may have been slightly different. Also it is possible that the abstraction level of our reconstruction process could omit some elements of [28]. We acknowledge that further tests with the reconstructed classifier are needed to adjust its architecture, and support our claims with stronger empirical evidence.

Construct validity. Our reconstruction of the feature set by Kurtanović and Maalej is not perfect, for we were unable to obtain the actual implementation. A manual comparison of the most informative features we obtained with those mentioned in [2], however, reveals a high degree of similarity.

External validity. Our attempt in identifying requirements datasets that represent heterogeneous industrial practices led to a varying performance across the datasets. 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 generalizability, we publicly share our code and data for further replications and studies.

VII. Related Work

Non-functional requirements. The NFR Framework [37], [38] is a cornerstone proposal for handling NFRs. This framework models requirements as goals and treats NFRs as *softgoals*, i.e., goals without a clear-cut criterion for satisfaction. In practice, however, early stage functional requirements can also be softgoals due to their vagueness and lack of detail [39]. Glinz [10] divides system requirements into functional requirements, attributes, and constraints. An attribute is either a performance requirement or a specific quality requirement. A NFR can be an attribute or a constraint of the system: NFRs are everything but the functional requirements. Eckhardt *et al.* [12] survey 11 requirements specifications to understand the nature of NFRs used in industry and discover that many NFRs include functionality, which is in parallel with the classification framework we use in this paper.

Automated Classifiers for RE. Zhang *et al.* [40] survey different ML techniques to automatically classify NFRs and conclude that individual words are the best index terms in text to indicate NFRs. Hussain *et al.* [31] use linguistic features such as cardinals, adverbs, and modals to train a classifier that identifies NFRs in software requirements specifications documents. The approach is also trained and tested on PROMISE. Singh and Sharme [41] combined automated identification and classification of requirements into non-functional requirement subclasses via a rule-based classification technique using thematic roles. They identified the priority of the extracted non-functional requirements according to their occurrence in multiple classes. Their application of this method to PROMISE resulted in F1-measure of 97%. Vogelsang and Winkler [7] introduce an approach to automatically classify the content elements of a natural language requirements specification document as "requirement" or "information" using convolutional neural networks (CNNs) with a high precision. Navarro-Almanza et al [42] used 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 used the PROMISE corpus in their evaluation and achieved precision, recall and f-measure values of 0.80, 0.785 and 0.77 respectively. Abad *et al.* [4] demonstrate that Binarized Naive Bayes performs the best when classifying NFRs into subclasses and preprocessing and unifying the requirements in the PROMISE dataset improves the classification performance. This result is not surprising since PROMISE includes requirements from 15 distinct projects written in different styles. Casamayor *et al.* [43] extract NFRs from natural language text. Mahmoud [44] associates key words with classes of NFRs, calculates co-occurrence of this terms and creates clusters using this metric. Then, the clusters are classified under sub-categories of NFRs with an average accuracy of 73%.

VIII. Conclusions and Future Work

This paper explores interpretable ML as a tool to build and evaluate classifiers in RE. We investigate the problem of distinguishing functional and quality aspects in requirements collections. Our higher-level feature set, applied to a ML classifier, leads to similar performance to the state of the art.

For interpretable ML to be effective, it is important to rely on a limited set of features that have clear semantics. We employ linguistic dependencies that define the main relationships in a sentence (see Fig. 5), as opposed to the low-level short sequences of words (n-grams) used by other researchers [2].

This paper presents a new approach for requirements classifiers, by describing a construction process and a set of features, rather than an off-the-shelf classifier. Future work should focus on constructing classifiers, as suggested in Sec. V-C, and on providing empirical evidence of their *in vivo* effectiveness.

ACKNOWLEDGMENT

We would like to thank the European Space Agency and the European Organisation for the Exploitation of Meteorological Satellites for sharing with us their requirements data sets.

REFERENCES

[1] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[2] Z. Kurtanović and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *IEEE International Requirements Engineering Conference (RE)*, 2017, pp. 490–495. [Online]. Available: https://doi.org/10.1109/RE.2017.82

[3] M. Binkhonain and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Systems with Applications*, 2019, accepted. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417419301459

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

[5] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[6] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-miner: Mining informative reviews for developers from mobile app marketplace," in *International Conference on Software Engineering*, 2014, pp. 767–778.

[7] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *IEEE International Requirements Engineering Conference Workshops*, 2016, pp. 39–45.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.

[9] Y. Ming, 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.

[10] M. Glinz, "On non-functional requirements," in *IEEE International Requirements Engineering Conference*, 2007, pp. 21–26.

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

[12] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional" requirements really non-functional? An investigation of non-functional requirements in practice," in *IEEE/ACM International Conference on Software Engineering*, 2016, pp. 832–842.

[13] The PROMISE repository of empirical software engineering data. [accessed 8-April-2019]. [Online]. Available: https://terapromise.csc.ncsu.edu/!/#repo/view/head/requirements/nfr

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

[15] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "Wonderweb deliverable D18, ontology library (final)," WonderWeb project, Tech. Rep., 2003. [Online]. Available: http://wonderweb.man.ac.uk/deliverables/documents/D18.pdf

[16] ESA Euclide Mission. [accessed 8-April-2019]. [Online]. Available: http://sci.esa.int/euclid/

[17] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: An incubator for cyber-physical systems research," in *International Conference on Software Engineering: NIER Track*, 2018, pp. 109–112.

[18] ReqView Example Requirements. https://www.reqview.com/doc/example-requirements-documents.html. [Online; accessed 8-April-2019].

[19] Leeds University Library Requirements. https://leedsunilibrary.files.wordpress.com/2013/06/repositoryfunctionalrequirementsv1-1_web_1_.xlsx. [Online; accessed 8-April-2019].

[20] Web Architectures for Services Platforms (WASP) Requirements. https://www.zenodo.org/record/581655. [Online; accessed 8-April-2019].

[21] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage publications, 2018.

[22] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "explainable-re/re-2019-materials," Jul. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3309669

[23] D. M. Berry, "Evaluation of tools for hairy requirements and software engineering tasks," in *IEEE International Requirements Engineering Conference Workshops*, 2017, pp. 284–291.

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

[25] J. A. Hanley 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.

[26] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, pp. 1–21, 2015.

[27] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *IEEE International Joint Conference on Neural Networks*, 2008, pp. 1322–1328.

[28] Z. Kurtanović and W. Maalej, "On user rationale in software engineering," *Requirements Engineering*, vol. 23, no. 3, pp. 357–379, 2018.

[29] MAST Applied Software Technology Group. [accessed 8-April-2019]. [Online]. Available: https://mast.informatik.uni-hamburg.de/app-review-analysis/

[30] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," in *Annual Meeting of the Association for Computational Linguistics: Volume 1, Long Papers*, 2018.

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

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

[33] SkopeRules. [accessed 8-April-2019]. [Online]. Available: https://skope-rules.readthedocs.io/en/latest/

[34] Universal Dependencies. [accessed 8-April-2019]. [Online]. Available: http://universaldependencies.org/

[35] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[36] A. Mavin, P. Wilksinson, S. Gregory, and E. Uusitalo, "Listens learned (8 lessons learned applying EARS)," in *IEEE International Requirements Engineering Conference*, 2016, pp. 276–282.

[37] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[38] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer, 2012.

[39] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens, "A more expressive softgoal conceptualization for quality requirements analysis," in *International Conference on Conceptual Modeling*, 2006, pp. 281–295.

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

[41] P. Singh, D. Singh, and A. Sharma, "Classification of non-functional requirements from SRS documents using thematic roles," in *IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, 2016, pp. 206–207.

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

[43] A. Casamayor, 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.

[44] A. Mahmoud, "An information theoretic approach for extracting and tracing non-functional requirements," in *IEEE International Requirements Engineering Conference*, 2015, pp. 36–45.