Empirical Software Engineering (2020) 25:5454–5497
https://doi.org/10.1007/s10664-020-09864-1

Automated demarcation of requirements in textual specifications: a machine learning-based approach

Sallam Abualhaija[1] · Chetan Arora[1,2] · Mehrdad Sabetzadeh[1,3] · Lionel C. Briand[1,3] · Michael Traynor[4]

Published online: 13 September 2020

Abstract

A simple but important task during the analysis of a textual requirements specification is to determine which statements in the specification represent requirements. In principle, by following suitable writing and markup conventions, one can provide an immediate and unequivocal demarcation of requirements at the time a specification is being developed. However, neither the presence nor a fully accurate enforcement of such conventions is guaranteed. The result is that, in many practical situations, analysts end up resorting to after-the-fact reviews for sifting requirements from other material in a requirements specification. This is both tedious and time-consuming. We propose an automated approach for demarcating requirements in free-form requirements specifications. The approach, which is based on machine learning, can be applied to a wide variety of specifications in different domains and with different writing styles. We train and evaluate our approach over an independently labeled dataset comprised of 33 industrial requirements specifications. Over this dataset, our approach yields an average precision of 81.2% and an average recall of 95.7%. Compared to simple baselines that demarcate requirements based on the presence of modal verbs and identifiers, our approach leads to an average gain of 16.4% in precision and 25.5% in recall. We collect and analyze expert feedback on the demarcations produced by our approach for industrial requirements specifications. The results indicate that experts find our approach useful and efficient in practice. We developed a prototype tool, named DemaRQ, in support of our approach. To facilitate replication, we make available to the research community this prototype tool alongside the non-proprietary portion of our training data.

Keywords Textual requirements · Requirements identification and classification · Machine learning · Natural language processing

This article belongs to the Topical Collection: *Requirements Engineering*

Communicated by: Kelly Blincoe, Daniela Damian, and Anna Perini

✉ Sallam Abualhaija
sallam.abualhaija@uni.lu

Extended author information available on the last page of the article.

1 Introduction

Requirements specifications (RSs) are arguably the most central artifacts to the requirements engineering process. An RS lays out the necessary characteristics, capabilities, and qualities of a system-to-be (van Lamsweerde 2009). RSs are typically intended for a diverse audience, e.g., users, analysts and developers. To facilitate comprehension and communication between stakeholders who have different backgrounds and expertise, RSs are predominantly written in natural language (Berry et al. 2003; Mich et al. 2004; Pohl 2010; Zhao et al. 2020).

The structure and content of a (textual) RS vary, depending on the requirements elaboration and documentation methods used. In general, an RS is expected to provide, in addition to the requirements, a variety of other information such as the system scope and environment, domain properties, concept definitions, rationale information, comments, and examples (van Lamsweerde 2009; Pohl 2010). A common problem during the analysis of an RS is telling apart the requirements from the other information.

Being able to distinguish requirements from the rest of an RS – that is, from non-requirements (Winkler and Vogelsang 2018) – is important for multiple reasons: First, requirements are typically the basis for development contracts (Arora et al. 2016). Making the requirements explicit helps avoid later disputes about contractual obligations. Second and from a quality standpoint, it is common to hold requirements to higher standards than non-requirements, considering the potentially serious implications of vagueness and ambiguity in the requirements (Berry et al. 2003; Pohl and Rupp 2011). Naturally, to be able to give extra scrutiny to the requirements, the analysts need to know where the requirements are located within a given RS. Finally, having the requirements explicated is essential for better supporting requirements verification and validation tasks, e.g., the specification of acceptance criteria and test cases, which are directly linked to the requirements (Pohl 2010).

The most immediate solution that comes to mind for distinguishing requirements from non-requirements in an RS is through leveraging the writing and markup conventions that may have been applied while writing the RS. Examples of these conventions include using modal verbs (e.g., “shall” and “will”) in requirements sentences and prefixing requirements with identifiers. Despite being simple and common in practice, such conventions do not lead to a reliable solution for recognizing between requirements and non-requirements. For instance, for an arbitrary given RS, one may neither know which conventions, if any, have been applied, nor be able to conclusively ascertain whether the conventions of choice have been applied consistently and unambiguously. Further, conventions per se do not protect against some important errors that may occur during requirements writing, e.g., the inadvertent introduction of new requirements while providing clarification or justification for other requirements.

Running Example. To illustrate, consider the example in Fig. 1. This example shows a small excerpt of a customer RS concerned with a space rover navigation system. To facilitate illustration, we have slightly altered the excerpt from its original form. The requirements in this excerpt, as identified by a domain expert, are the segments marked R1–R7 and shaded green. The non-requirements are marked N1–N5. As we argue next, one cannot accurately recognize the requirements in this excerpt through applying simple heuristics.

The intuition of selecting as requirements only sentences with modal verbs is not good enough, because some requirements, e.g., R4, R6, and R7, have none. Further, non-requirements too may contain modal verbs, e.g., "will" in N5. Similarly, selecting as requirements only sentences prefixed with alphanumeric patterns can be inaccurate, since these patterns may not have been used consistently or may be overlapping with other structural elements, e.g., section headings. In our example, R1, R4, R6, and R7 have unique alphanumeric identifiers, but the numbering scheme is the same as that used for sectioning the text. At the same time, R2 and R3 are items in a simple enumerated list; and, R5 has no numbering at all. Another simple heuristic, namely filtering out segments marked as supplementary material via such cue phrases as "Note:", can lead to both missed requirements – in our example, R5 – as well as numerous false positives – in our example, N1–N4.

As the example of Fig. 1 highlights, the seemingly simple task of separating requirements from non-requirements in an RS cannot be addressed accurately through easy-to-automate heuristics. Doing the task entirely manually is not an attractive alternative due to being very tedious and time-consuming.

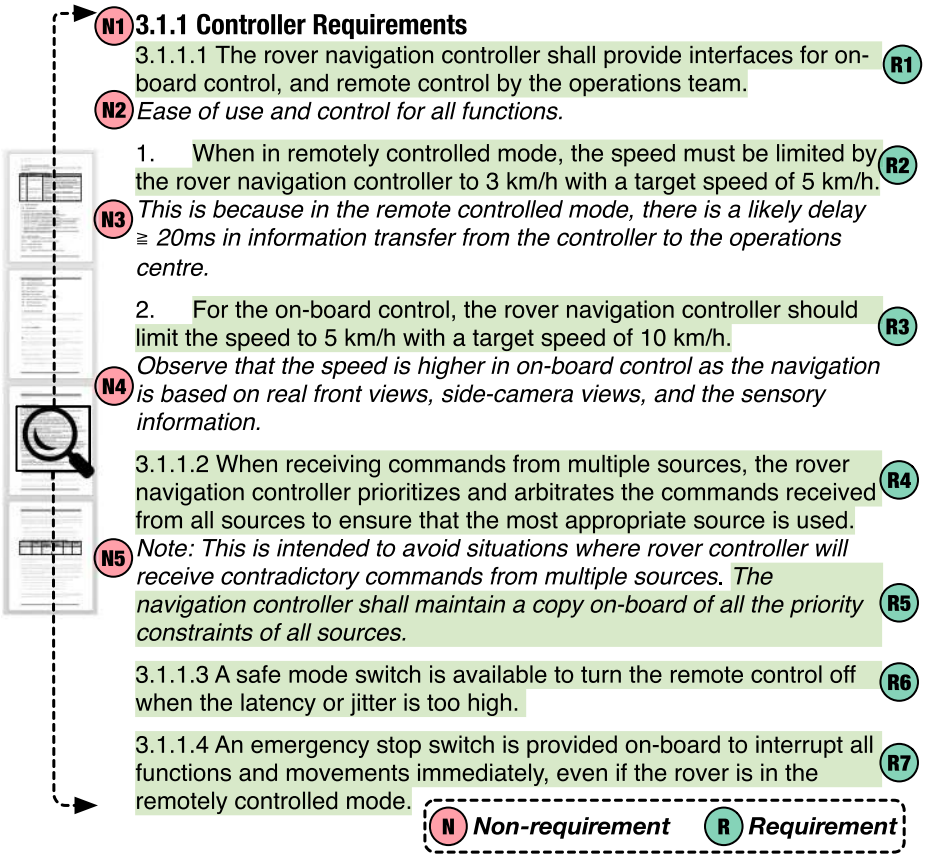


Fig. 1 Excerpt from a Textual Requirements Specification

Usage Scenarios. In this article, we develop a practical, accurate, and fully automated approach for *requirements demarcation* in textual RSs. By requirements demarcation, we mean recognizing and precisely delimiting the text segments that represent requirements and non-requirements. Our approach is in direct response to an industrial need presented to us by a leading requirements tool vendor and our collaborating partner, QRA Corp (https://qracorp.com). An internal market study by QRA suggests that practitioners frequently have to review large RSs, particularly customer-provided ones, in search of requirements. Some common situations necessitating such reviews are as follows: (1) A company wishing to respond to a complex call for tender has to draw up an informed estimate about development costs, usually under tight time constraints. Doing so usually entails going through various unstructured or loosely structured tender documents containing both user requirements and supplementary information (Falkner et al. 2019). The ability to quickly and accurately spot the requirements in tender documents is key to arriving at competitive and yet profitable bids; (2) In many contexts, requirements are contractually binding. The stakeholders thus need to systematically negotiate the requirements and define the obligations for each party; this naturally leads to the need for making the requirements statements in RSs explicit; and (3) One may want to import an unstructured or loosely structured RS into a requirements management (RM) tool such as IBM DOORS (2020). In such a case, knowing where are the requirements statements is a direct prerequisite for document segmentation and transitioning to a structured format. In light of the above usage scenarios, our collaborating partner has shown interest in a technology that would help requirements engineers more efficiently identify the requirements in complex textual specifications.

Much work has been done in the RE community on automating the identification and classification of requirements-related information in textual documents (Winkler and Vogelsang 2018; Falkner et al. 2019). However, and as we argue more precisely in Section 6, none of the existing strands of work lead to a general-purpose requirements demarcation solution. The novelty of our approach is in simultaneously (1) being domain- and terminology-independent, (2) requiring no user input about the structure or content of the RS needing to be processed, and (3) placing no restrictions on the types of requirements that can be identified. Achieving these characteristics is paramount in our context, noting that the ultimate goal of our partner is to offer a *push-button* requirements demarcation technology that can be used by a diverse set of clients and over RSs written in a variety of styles.

Contributions. We employ machine learning (ML) to devise an automated approach for requirements demarcation. Our ML classification model is based on generic features that can be applied to a wide range of RSs irrespective of the specific template, terminology and style used. Our ML features take into consideration some common conventions that may have been applied during RS writing (illustrated in Fig. 1). However, the features do not take the presence or consistent application of these conventions for granted. Instead, the features introduce several more advanced criteria based on the syntactic and semantic properties of textual statements. These additional criteria lead to a more accurate classification of requirements and non-requirements than when conventions are considered alone. To account for these criteria, our approach relies on information extraction via natural language processing (NLP).

Our approach focuses exclusively on free-form (unrestricted) RSs; we do not address structured RS types such as use-case descriptions and user stories. This decision was motivated by the prevalence of free-form RSs in practice (Mich et al. 2004). While our approach is not limited to free-form RSs, it is not necessarily optimized for structured RSs, since it does not envisage ML features for picking up on any specific structural restrictions. This said, we believe requirements demarcation is less likely to need an ML-based solution when RSs are sufficiently structured; for such RSs, the structure in itself is often sufficient for a rule-based identification of requirements and non-requirements.

We empirically evaluate our approach using a dataset of 33 industrial RSs from 14 different application domains and originating from 20 different organizations. These RSs, which were labeled either directly by industry experts or by an independent, trained annotator (non-author), collectively contain 8046 requirements and 12673 non-requirements. We exploit this dataset following standard procedures for ML algorithm selection and tuning, training, and testing. When applied to an RS, no portion of which has been exposed during training, our approach yields an average precision of 81.2% and average recall of 95.7%. In comparison, our approach leads to an average improvement of 16.4% in precision and 25.5% in recall vis-à-vis simple baselines that recognize requirements based on the presence of modal verbs and requirements identifiers. Our evaluation further examines the execution time of our approach, showing that the approach is practical for both batch analysis and interactive demarcation. To study the practical usefulness of our approach, we elicit expert feedback from four practitioners on RSs demarcated by the approach. The results indicate that the experts in our study have a positive opinion about the efficacy of our requirements demarcation approach. The experts further recognize that our approach brings to their attention important information that they may otherwise overlook. While promising, the results of our expert interview surveys are only indicative. More studies are required in the future to more definitively assess the usefulness of our approach in practice.

We developed a tool, named DemaRQ, which implements our requirements demarcation approach. DemaRQ enables users to directly identify the requirements in an input RS using the best ML classification model emerging from our empirical evaluation. This tool, alongside the non-proprietary RSs we use for training, are publicly available at https://sites.google.com/view/demarq.

This article is an extension of a previous conference paper (Abualhaija et al. 2019) published at the 27th IEEE International Requirements Engineering conference (RE'19). The current article enhances both the technical aspects and the empirical evaluation of our approach, compared to the conference version. Specifically, we introduce a post-processing step aimed at improving the accuracy of requirements demarcation; this is discussed in Section 3. We complement our evaluation dataset with three additional RSs, and perform surveys with industry experts on these RSs; this is discussed in Section 5.2.2. The surveys provide insights about how practicing engineers perceive of the usefulness of our approach for requirements demarcation. Finally, we provide, in Section 4, an in-depth technical description of our prototype tool.

Structure. Section 2 provides background. Section 3 presents our approach. Section 4 discusses tool support. Section 5 describes our empirical evaluation. Section 6 compares with related work. Section 7 discusses threats to validity. Section 8 concludes the article.

2 Background

In this section, we review the machine learning and natural language processing background for our approach.

2.1 Machine Learning

Our approach is based on *supervised* ML, meaning that it requires labeled data for training. Our labeled data is made up of RSs whose different segments have been marked by human experts as being requirements or non-requirements. Supervised techniques are categorized into *classification* and *regression* (Goodfellow et al. 2016); the former is aimed at predicting categorical outputs, and the latter – at predicting real-valued outputs. What we are concerned with, namely distinguishing between requirements and non-requirements, is a binary classification problem. In our evaluation (Section 5), we empirically examine several alternative ML classification algorithms in order to determine which one is the most accurate for our purpose.

ML classification algorithms typically attempt to minimize misclassification, giving equal treatment to different types of misclassification. That is, in a binary classification problem with classes A and B, misclassification of an element of class A as B or the misclassification of an element of class B as A are treated equally. However, in many problems, like ours, the costs associated with the two misclassification types are not symmetric. For instance, in medical diagnosis, the cost of falsely misdiagnosing a healthy person (false positive) could be an unnecessary escalation, whereas the cost of falsely misdiagnosing as healthy an actual patient (false negative) is postponed or failed treatment. Clearly, the cost of two misclassifications is not symmetrical. Similarly, the cost of misclassifying a non-requirement as a requirement (false positive) is considerably less than that of misclassifying a requirement as a non-requirement (false negative). The rationale here is that, as long as false positives are not too many, the effort of manually discarding them is a compelling trade-off for better recall, i.e., the ability to identify all the requirements (Berry 2017).

ML can be tuned to take account of misclassification costs either (1) *during* training or (2) *after* training. The former strategy is known as *cost-sensitive learning* and the latter – as *cost-sensitive classification* (Goodfellow et al. 2016). In our approach, we employ *cost-sensitive learning*, which is generally considered to be the more effective of the two (Witten et al. 2016). We note that cost-sensitive learning may also be used for addressing class imbalance: the situation where data instances for certain classes are much more prevalent than for others. Neither of our classes – requirement and non-requirement – are under-represented in our dataset (see Section 5). We use cost-sensitive learning exclusively for mitigating, as much as possible, false negatives.

In Section 5, we experiment with several ML classification algorithms to determine the most accurate alternative for our problem. We further discuss the effectiveness of using cost-sensitive learning, when we consider the costs associated with false positives (FPs) and false negatives (FNs) to be different.

The process of designing features based on the characteristics that distinguish different classes in a classification problem is called *feature engineering* (Manning et al. 2008). For text classification, this process involves human craft and of course requires knowledge

about the problem at hand. There are also automated feature generation methods reported in the state of the art for text classification. However, these methods rely heavily on the terminology in the documents, e.g., using the Bag Of Words (BOW) model (Aggarwal 2018), and are thus domain-dependent. We elaborate more on how we designed our domain- and terminology-independent features set in Section 3.

2.2 Natural Language Processing

Natural language processing (NLP) is concerned with the computerized understanding, analysis, and production of human-language content (Jurafsky and Martin 2009; Indurkhya and Damerau 2010). In this article, we employ three main NLP technologies: (1) *constituency parsing* for delineating the structural units of sentences, notably Noun Phrases (NPs) and Verb Phrases (VPs), (2) *dependency parsing* for inferring grammatical dependencies between the words in sentences, e.g., subject-object relations, and (3) *semantic parsing* for building a representation of the meaning of a sentence based on the meaning of the sentence's constituents.

Semantic parsing relies on lexical resources such as WordNet (Miller 1995) for information about words' senses (meanings) and the relations between these senses. In this article, our use of semantic parsing is limited to distinguishing different sentences based on the semantic category of their verbs. For this purpose, we use WordNet's categorization of verbs (Princeton University 2010). We are specifically interested in the following verb classes:

(1) *cognition*, referring to verbs that describe reasoning or intention, e.g., "think", "analyze" and "believe"; (2) *action*, referring to verbs that describe a motion or change, e.g., "go" and "fall", and (3) *stative*, referring to verbs that describe the state of being, e.g., "is" and "has". Including cognition verbs is motivated by recent work on argumentation mining (Habernal et al. 2014; Habernal and Gurevych 2017) and the relevant applications on mining user opinions from products and software reviews (Kurtanovic and Maalej 2017b). It is common to use cognition verbs in an argumentative context (Fetzer and Johansson 2010). In the RE field, Kurtanovic and Maalej (2017b) mine user reviews for explicit arguments by relying on the syntactic structure of sentences in which there exist purpose or reason clauses. **Following this earlier work, we use cognition verbs to capture the sentences that provide rationale or reasoning around requirements.** Including action and stative verbs is inspired by Rolland and Proix (1992) who used Fillmore's basic English verb types (Cook 1989) (in addition to other information) for creating conceptual specifications from NL statements.

NLP-based analysis is typically performed using a pipeline of NLP modules. Fig. 2 shows the pipeline we use in our work. The first module in the preprocessing group of modules is the *Tokenizer*, which breaks the input text – in our case, an RS – into tokens. The tokens may be words, numbers, punctuation marks, or symbols. The second preprocessing module, the *Sentence Splitter*, splits the text into sentences based on conventional delimiters, e.g., period. The third preprocessing module, the *POS Tagger*, assigns part-of-speech (POS) tags, e.g., noun, verb, and adjective, to each token. The results from the preprocessing steps constitute the input to the three parsing techniques, outlined earlier. The output of

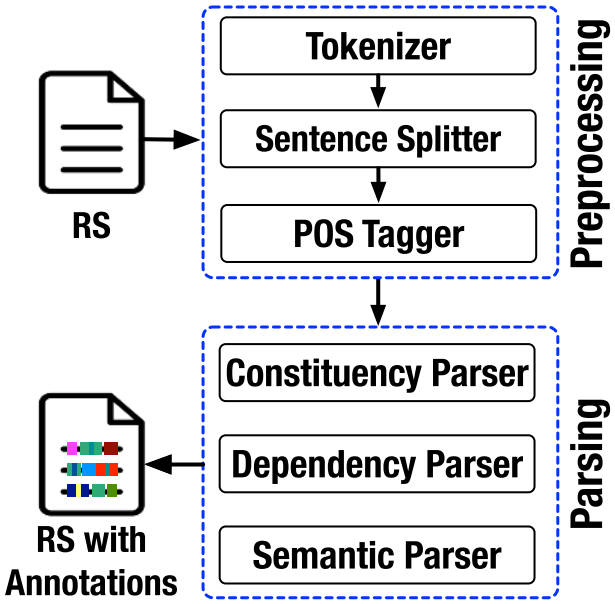


Fig. 2 Our NLP pipeline

the NLP pipeline is a set of annotations (metadata) attached to the elements within the input RS. These annotations are exploited by our approach as we explain next.

3 Approach

Figure 3 presents an overview of our approach. The input to the approach is a textual RS. We treat each RS as a collection of *sentences*. By sentence, we do not necessarily mean a grammatical sentence, but rather a text segment that has been marked as a "sentence" by the Sentence Splitter module of the NLP pipeline in Fig. 2. According to this definition, the excerpt of Fig. 1 has 12 sentences: N1–N5 and R1–R7. Our unit of classification is a sentence as per the above definition.

As mentioned earlier, the classification is binary with the two classes being requirement and non-requirement. In the remainder of the article, given the binary nature of classification, we refer to each sentence as a *requirement candidate*. The output of the approach is a demarcated RS; this is the input RS where each requirement candidate has been marked as either a requirement or a non-requirement.

The approach works in four phases, labeled 1–4 in Fig. 3. In the first phase, we run the NLP pipeline of Fig. 2 on the input RS and further derive certain document-level metadata related to frequencies, e.g., the distribution of different modal verbs in the RS. In the second phase, we construct a feature matrix for classification, using the NLP results and the

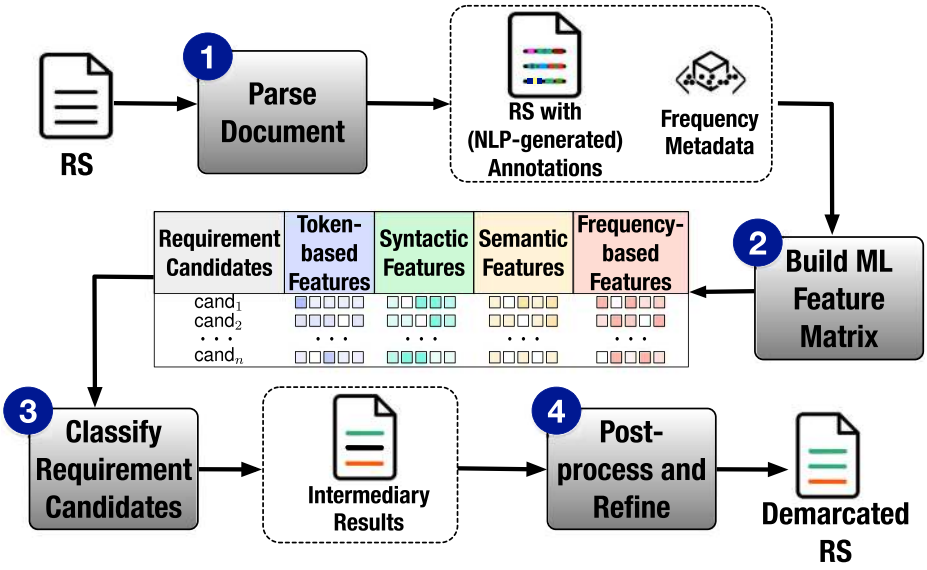


Fig. 3 Approach Overview

frequency information obtained from the first phase. In the third phase, we perform the classification using ML. In the fourth and final phase, we process and refine the classification results through post-processing. The rest of this section elaborates the four phases of our approach.

3.1 Parsing the Requirements Specification

The first phase of our approach is to execute the NLP pipeline of Fig. 2. The Sentence Splitter module of the pipeline yields the requirement candidates, with the other modules in the pipeline providing part-of-speech, syntactic and semantic annotations for the requirement candidates. Upon the completion of the execution of the NLP pipeline, we compute the following frequency-related metadata using the NLP-generated annotations: (1) the most frequent modal verb in the RS, (2) the top 1% most frequent noun phrases, and (3) frequency levels for the different identifier patterns used within the RS.

To infer the identifier patterns (id patterns, for short), we first extract all the alphanumerics in the RS. We then replace in each alphanumeric any sequence of digits with *d*, any lower-case character with *c*, and any upper case character with *C*. The id patterns are the distinct shapes obtained from this replacement. For instance, in our example of Fig. 1, we infer three id patterns: *"d.d.d"* from N1, *"d.d.d.d"* from R1, R4, R6, and R7 and *"d."* from R2 and R3. We count the occurrences of each pattern within the entire RS. We then ascribe to each pattern one of the following frequency levels: *high*, *medium*, *low*. A pattern has high frequency if its number of occurrences divided by the total number of occurrences of all the patterns falls within $[\frac{2}{3}, 1]$. A pattern has medium frequency if the above ratio is within $[\frac{1}{3}, \frac{2}{3})$, and low frequency if the ratio is within $(0, \frac{1}{3})$. We note that the exact shapes of the id patterns are of no interest to us, considering that these shapes invariably differ across different RSs. We are interested in the id patterns and their frequencies due to the following general intuition: A requirement candidate containing an occurrence of a frequent id pattern

is more likely to be a requirement than a non-requirement. A precise realization is presented later (see *idPatternFrequency* in Table 1).

The output of the parsing phase is as follows: (1) the requirement candidates representing the units to classify, (2) NLP annotations for the elements of individual requirement candidates , and (3) certain RS-wide frequency metadata as discussed above. This output is used for building a feature matrix (Phase 2 of Fig. 3).

3.2 Building an ML Feature Matrix

Feature Engineering. Our feature design has been driven by the need to keep the features generic, i.e., the features should not rely on the domain, terminology, content, or formatting of any particular RS. To meet this criterion, we orient our features around structural and semantic properties that are meaningful irrespective of the exact characteristics of an individual RS. Following the practices of designing linguistic and stylistic features in NLP applications (Stamatatos 2009), we designed our features in an iterative manner. The first author, who has a strong background in linguistics, analyzed three RSs from different domains, and created the first set of features. This set was then refined in an iterative manner.

Features. Table 1 lists our features followed by an illustrating example based on the excerpt in Fig. 1. These features are computed for every requirement candidate. The table is divided into four sub-tables, each of which represents a feature category as we elaborate next. For each feature, the table provides an id, short name, type, description, and an intuition. We organize the features into four categories:

- The *token-based features* (Tok1–Tok6) are based on the token-level information in a requirement candidate.
- The *syntactic features* (Syn1–Syn8) are derived from the syntax-related information in a requirement candidate, e.g., POS tags, phrasal structure, and grammatical dependency relations.
- The *semantic features* (Sem1–Sem3) are derived from the semantic categories of the verbs in a requirement candidate. These categories were defined in Section 2.2.
- The *frequency-based features* (Frq1–Frq3) are derived for each cand based on the document-wide frequency metadata discussed in Section 3.1 as well as the syntax-related information within that particular requirement candidate.

The output of this phase is a feature matrix where the rows represent the requirement candidates within the input RS and where the columns represent the 20 features listed in Table 1.

Example (Feature Computations) Below, we exemplify our proposed features using the excerpt shown in Fig. 1. To facilitate explanation, we have selected from Fig. 1 some of the requirement candidates and repeated them in the same order as they appear in that figure. For each selected candidate, we explain the relevant features.

N1 3.1.1 Controller Requirements

Tok1: numTokens = 7; because the tokenizer returns 7 tokens: {“3”, “.”, “1”, “.”, “1”, “Controller”, “Requirements”}.

Tok2: numAlphabetics = 2; because there are only two alphabetic tokens, namely “Controller” and “Requirements”.

Tok3: numOneCharTokens = 5; as per the results of the tokenizer above.

Tok4: startsWithId = TRUE; because N1 starts with the id: “3.1.1”.

Table 1 Features for learning

ID	Feature (Type)	Description (**D**) & Intuition (**I**)
(a) Token-based features		
Tok1	numTokens (Numeric)	**(D)** Number of tokens in a requirement candidate. **(I)** A requirement candidate that is too long or too short could indicate a non-requirement.
Tok2	numAlphabetics (Numeric)	**(D)** Number of alphabetic words in a requirement candidate. **(I)** Few alphabetic words in a requirement candidate could indicate a non-requirement.
Tok3	numOneCharTokens (Numeric)	**(D)** Number of one-character tokens in a requirement candidate. **(I)** Too many one-character tokens in a requirement candidate could indicate a non-requirement, e.g., section headings.
Tok4	startsWithId (Boolean)	**(D)** TRUE if a requirement candidate starts with an alphanumeric segment containing special characters such as periods and hyphens, otherwise FALSE. **(I)** Alphanumeric segments with special characters could indicate identifiers for requirements.
Tok5	startsWithTriggerWord (Boolean)	**(D)** TRUE if a requirement candidate begins with a trigger word ("Note", "Rationale", "Comment"), otherwise FALSE. **(I)** A trigger word at the beginning of a requirement candidate is a strong indicator for a non-requirement.
Tok6	hasUnits (Boolean)	**(D)** TRUE if a requirement candidate contains some measurement unit, otherwise FALSE. **(I)** According to several domain experts consulted throughout our work, the presence of measurement units increases the likelihood of a requirement candidate being a requirement.
(b) Syntactic Features		
Syn1	hasVerb (Boolean)	**(D)** TRUE if a requirement candidate has a verb per POS tags, otherwise FALSE. **(I)** A requirement candidate without a verb is unlikely to be a requirement.
Syn2	hasModalVerb (Boolean)	**(D)** TRUE if a requirement candidate has a modal verb, otherwise FALSE. **(I)** The presence of a modal verb is a good indicator for a requirement candidate being a requirement.
Syn3	hasNPModelVP (Boolean)	**(D)** TRUE if a requirement candidate contains a sequence composed of a Noun Phrase (NP) followed by a Verb Phase (VP) that includes a modal verb, otherwise FALSE. **(I)** The intuition is the same as that for Syn2. Syn3 goes beyond Syn2 by capturing the presence of the NP preceding a modal VP. This NP typically acts as a subject for the VP.
Syn4	startsWithDetVerb (Boolean)	**(D)** TRUE if a requirement candidate, excluding head alphanumeric patterns / trigger words, begins with a pronoun or determiner followed by a verb, otherwise FALSE. **(I)** This is a common natural-language construct for justification and explanation, and thus could indicate a non-requirement.
Syn5	hasConditionals (Boolean)	**(D)** TRUE if a requirement candidate has a conditional clause, otherwise FALSE. **(I)** Conditional clauses are more likely to appear in requirements than non-requirements.

Table 1 (continued)

ID	Feature (Type)	Description (**D**) & Intuition (**I**)
Syn6	hasPassiveVoice (Boolean)	**(D)** TRUE if a requirement candidate has passive voice through some dependency relation, otherwise FALSE. **(I)** Requirements not containing modal verbs may be specified in passive voice.
Syn7	hasVBToBeAdj (Boolean)	**(D)** TRUE if, in requirement candidate, there is some form of the verb "to be" appearing as root verb followed by an adjective, otherwise FALSE. **(I)** The pattern described is more likely to appear in requirements.
Syn8	isPresentTense (Boolean)	**(D)** TRUE if a requirement candidate has some root verb which is in present tense, otherwise FALSE. **(I)** Sometimes, requirements are written in present tense rather than with modal verbs.
(c) Semantic Features		
Sem1	hasCognitionVerb (Boolean)	**(D)** TRUE if a requirement candidate has some verb conveying reasoning or intention, otherwise FALSE. **(I)** Reasoning and intention are a common characteristic for non-requirements.
Sem2	hasActionVerb (Boolean)	**(D)** TRUE if a requirement candidate has some verb conveying motion or change of status, otherwise FALSE. **(I)** Action verbs are common in requirements for describing behaviors and state changes.
Sem3	hasStativeVerb (Boolean)	**(D)** TRUE if a requirement candidate has some stative verb, otherwise FALSE. **(I)** Stative verbs are common in requirements for describing system properties.
(d) Frequency-based Features		
Frq1	idPatternFrequency (Enumeration)	**(D)** Maximum frequency level (high, medium, low) associated with the identifier pattern with which a given requirement candidate starts. If a requirement candidate does not start with an alphanumeric pattern, the returned value is NA (not applicable). **(I)** A frequent id pattern in a requirement candidate is likely to signify a requirement. This is because alphanumeric are prevalently used for marking requirements.
Frq2	hasMFModalVerb (Boolean)	**(D)** TRUE if a requirement candidate contains the most frequent modal verb of the RS, otherwise FALSE. **(I)** While a consistent application of modal verbs cannot be guaranteed, the most frequent modal verb is a strong indicator for requirements.
Frq3	hasHFNP (Boolean)	**(D)** TRUE if a requirement candidate contains a highly frequent (top 1%) NP in the RS, otherwise FALSE. **(I)** Highly frequent NPs (after stopword removal) often signify core concepts, e.g., the system and its main components. These concepts are more likely to appear in requirements.

R3 2. For the on-board control, the rover navigation controller should limit the speed to 5 km/h with a target speed of 10 km/h.

Tok6: hasUnits = TRUE; because R3 contains the unit: "km/h".
Syn1: hasVerb = TRUE; because R3 contains a verb.
Syn2: hasModalVerb = TRUE; because of the modal verb: "should".
Syn3: hasNPModalVP = TRUE; because the NP-modal-VP has a match in R3: "The rover navigation controller should limit".
Sem2: hasActionVerb = TRUE; because of the verb: "limit".
Frq3: hasHFNP = TRUE; because R3 contains the most frequent NP in Fig. 1 which is: "rover navigation controller".
Frq1: idPatternFrequency = low; according to the id frequencies in Fig. 1.
Frq2: hasMFModalVerb = FALSE; because "shall" is the most frequent modal verb in Fig. 1.

R4 3.1.1.2 When receiving commands from multiple sources, the rover navigation controller prioritizes and arbitrates the commands received from all sources to ensure that the most appropriate source is used.

Syn5: hasConditionals = TRUE; because R4 has a conditional clause: "When receiving commands from multiple sources".
Syn8: isPresentTense = TRUE; because of the root verb: "prioritizes".
Frq1: idPatternFrequency = medium; according to the id frequencies in Fig. 1.

N5 Note: This is intended to avoid situations where rover coltroller will recieve contradictory commands from multiple sources.

Tok5: startsWithTriggerWord = TRUE; because N5 starts with: "Note".
Syn4: startsWithDetVerb = TRUE; because when the trigger word "Note:" is excluded from the head of N5, the remainder begins with a determiner followed by a verb: "This is".
Sem1: hasCognitionVerb = TRUE; because of the verb: "intended".

R5 The navigation controller shall maintain a copy on-board of all the priority constraints of all sources.

Sem3: hasStativeVerb = TRUE; because of the verb: "maintain".
Frq1: idPatternFrequency = NA; because R5 has no id.

R6 3.1.1.3 A safe mode switch is available to turn the remote control off when the latency or jitter is too high.

Syn7: hasVBToBeAdj = TRUE; because R6 includes: "is available".

R7 3.1.1.4 An emergency stop switch is provided on-board to interrupt all functions and movements immediately, even if the rover is in the remotely controlled mode.

Syn6: hasPassiveVoice = TRUE; because R7 contains: "is provided".

3.3 Classifying Requirements and Non-requirements

Classification is done by applying a pre-trained classification model to the feature matrix from the second phase of our approach. The output of classification is a demarcation of the requirements and non-requirements in the input RS.

For training, we use a collection of industrial RSs (see Section 5). The training set was created by (1) individually subjecting each RS in the collection to the first and second phases of our approach, (2) having independent experts manually label the requirement candidates in each RS, and (3) merging the (labeled) data from the individual RSs into a single (training) set.

Selecting the most effective ML classification algorithm and tuning it are addressed by the first research question (RQ1) in our evaluation, presented in Section 5.

3.4 Post-processing and Refinement of the Results

The classifier in Step 3 predicts for each requirement candidate whether it is likely to be a requirement or non-requirement. In this post-processing step, we refine the results of the classifier by performing *list detection* as described next.

In the list detection refinement, we parse the textual RS and look for any list environment, which usually contains two components: a header and sub-items. In most cases, the sub-items are short phrases that do not constitute a grammatical sentence. Nevertheless, both headers and sub-items are considered as separate sentences in our approach because the unit of analysis is the sentence given by the NLP Sentence Splitter module.

List detection marks the beginning and the end of a list environment and, with this information, it is possible to connect the different sub-items to their corresponding header component. The intuition behind this post-processing step is to reduce the number of missed (parts of) requirements. Our post-processing step works as follows: if the header of a list environment has been marked as a requirement by the classifier, then all of the connected sub-items of that list environment are also marked as requirements.

While this post-processing step can reduce the number of false negatives, the step can potentially lead to an increase in the number of false positives. For example, when the header of a list is a false positive, including the list's sub-items is likely to introduce additional false positives. Despite this potential trade-off, our experiments, conducted for this article, suggest that list environments are more common within requirements than non-requirements. This makes the post-processing step worthwhile, especially considering the importance of high recall in our context.

4 Tool Support

We implemented our approach into a tool named "**Dema**rcator for **R**e**Q**uirements" (DemaRQ). Our tool has four main components corresponding to the steps in our approach (see Fig. 3). The tool architecture is depicted in Fig. 4. Our implementation builds on the Apache UIMA framework.[1]

(1) ***Document parsing.*** The first component expects an RS as an input, as explained in Section 3. All the input RSs we used in this article were either PDF or MS-Word files. We transformed the PDF ones into MS-Word using Adobe Acrobat Export PDF. The MS-Word reader in this component can be re-implemented for other types of textual documents. The different components can be easily plugged in or out of our pipeline as needed thanks to the flexibility of UIMA. In this component, we have two main methods, namely: extract_sentences and compute_frequency_metadata.

[1] https://uima.apache.org/

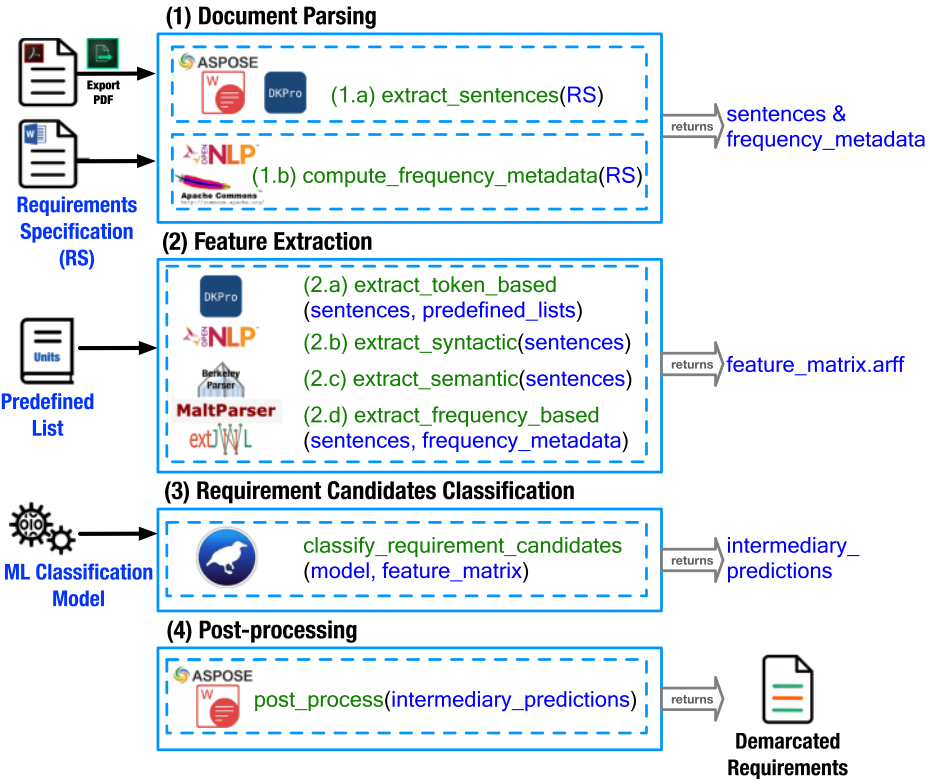


Fig. 4 Tool overview

We extract the text from the input MS-Word RS using Aspose API (AsposeWords 2018). In particular, we use Aspose.Words for Java, since our tool is Java-based. Then, we run the *LanguageToolSegmenter* provided in the DKPro toolkit (Eckart de Castilho and Gurevych 2014) for tokenization and sentence splitting to get the sentences from the text. While parsing the document, we also compute the frequency metadata that will be needed to extract the frequency-based features in the next component. Two frequency-based features require some syntactic information as follows. Computing *hasMFModalVerb* requires the POS-tags of the words, and *hasHFNP* requires the phrasal analysis of the sentences produced by a text chunker. For both syntactic analyses, we use OpenNLP (Apache OpenNLP 2017). The last feature, *idPatternFrequency*, requires the tokenization results only. This component returns a list of sentences representing the input RS and the frequency metadata.

(2) ***Feature extraction.*** The second component is computing the different categories of our features for each requirement candidate (that corresponds to a sentence) in the input RS. The token-based features require the results of the tokenizer only. One token-based feature (*hasUnits*) uses a predefined list, namely the list of measurement units (Cunningham et al. 2013). We use DictionaryAnnotator provided in DKPro for looking up the measurements list in the sentences. For computing the syntactic features, in addition to the POS-tagger and text chunker, we need the constituency

parser for *hasConditionals* and the dependency parser for *hasPassiveVoice*, *hasVBToBeAdj*, and *isPresentTense*. Our syntactic pipeline employs Berkeley (Petrov et al. 2006) for constituency parsing and Malt (Nivre et al. 2006) for dependency parsing. Semantic features require information about the verb categories. We extract this information from the lexical file names provided in WordNet using the Java-based API: extJWN (Walenz and Didion 2011). The first frequency-based feature (*idPatternFrequency*) is computed as explained in Section 3 using the frequency metadata created for all id patterns in the first component (Document Parsing). The two remaining frequency-based features require the same prior operational pipeline as the one used for computing the frequency metadata in the previous component: tokenizer, POS-tagger, and chunker. For each requirement candidate, we compare the syntactic information extracted from this particular requirement candidate with the frequency metadata from the previous component. Note that the latter two frequency-based features (*hasMFModalVerb*, *hasHFNP*) are Boolean. This component returns a feature matrix containing the computed feature vectors with unknown predicted classes (denoted as "?") for each requirement candidate in the RS. The feature matrix is saved in an ARFF file (Witten et al. 2016) to be used as input for the classification step, explained next.

(3) ***Requirement candidates classification.*** This component expects a pre-trained model (Random Forest with cost sensitive learning in our case) and an ARFF file created in the previous component. We use the Java-based WEKA API (Witten et al. 2016) for both creating the pre-trained model as well as classifying the requirement candidates into requirements or non-requirements. The results of this component is then the same ARFF file where the unknown class values are replaced with the predicted class for each requirement candidate ("Requirement" or "Non-requirement").

(4) ***Post-processing.*** In this component, we first detect list environments in the original RS. If the list environment corresponds to an automatically generated list by MS-Word, then this can be detected using the Aspose API during parsing. Otherwise, we detect list environments as explained in Section 3. Table 2 provides a summary of the used APIs. The names of the APIs are text anchors that lead to the relevant websites.

5 Evaluation

In this section, we empirically evaluate our requirements demarcation approach.

Table 2 The APIs and libraries in our tool

Task	Library
Document parsing	Aspose.Word for Java
Tokenization	LanguageToolSegmenter
POS-Tagging	Apache OpenNLP
Text chunking	Apache OpenNLP
Constituency parsing	Berkeley Parser
Dependency parsing	Malt Parser
Semantic parsing	extJWNL
ML-based classification	WEKA

5.1 Research Questions (RQs)

Our evaluation aims to answer the following RQs:

RQ1. ***Which ML classification algorithm yields the most accurate results?*** Our requirements demarcation approach can be implemented using several alternative ML classification algorithms. RQ1 identifies the most accurate alternative. We use a *z-test* (Dietterich 1998) to test differences in proportions and verify which ML-classification algorithm performs significantly better than the others.

RQ2. ***What are the most influential ML features for requirements demarcation?*** Table 1 listed our features for learning. RQ2 examines and provides insights about which of these features are the most important for requirements demarcation.

RQ3. ***How effective is our approach for requirements demarcation on previously unseen RSs?*** Building on the result of RQ1, RQ3 assesses the accuracy of our approach over previously unseen RSs.

RQ4. ***What is the execution time of our approach?*** To be applicable, our approach should run within practical time. RQ4 investigates whether this is the case.

RQ5. ***Is there any good tradeoff for reducing execution time without a major negative impact on demarcation quality?*** In an interactive mode where analysts submit new RSs or change them and want immediate feedback, response time is a key consideration. In RQ5, we look into the tradeoff between the cost of computing different features and the benefits gained in return. This enables us to determine whether we can leave out some expensive-to-compute features without significantly degrading the quality of demarcation.

RQ6. ***Do practitioners find our approach useful in practice?*** For our approach to be adopted in practice, the practitioners need to find the requirements demarcation results produced by our approach useful. RQ6 aims to analyze the practitioners' perspective on expected benefits.

5.2 Data Collection Procedure

Our data collection focused on procuring a representative set of free-form RSs and demarcating these RSs. We gathered 33 industrial RSs from 14 application domains, including, among others, information systems, automotive, healthcare, aviation, aerospace, shipping, telecommunications, and networks. These RSs originate from a total of 20 different organizations.

Among the 33 RSs, 30 RSs were manually demarcated for training the ML classification model in our approach (Step 3 of Fig. 3), and for answering RQ1–RQ5. The data collection procedure details for manual demarcation are presented in Section 5.2.1. The remaining three RSs were demarcated automatically using our approach in order to address RQ6. A panel of experts from two different industry partners reviewed these automatically demarcated RSs as a part of three interview surveys. The overall survey protocol is covered in Section 5.2.2.

5.2.1 ML Data Curation and Preparation

Among the 30 manually demarcated RSs, 12 had their requirements already marked by our industry partner in collaboration with the respective system clients. The requirements

in the remaining 18 RSs were marked by a paid, professional annotator (non-author). The annotator's background is in linguistics with a specialization in English writing and literature. Before starting her work on these 18 RSs, the annotator received two half-day courses on requirements specification by one of the researchers (an author). Anticipating that some statements in the RS would not be conclusively classifiable as a requirement or non-requirement, the researchers explicitly instructed the annotator to favor the requirement class whenever she was in doubt as to whether a statement was a requirement or non-requirement. This decision was motivated by the need to minimize missed requirements (false negatives) in our automated solution. In addition to training, the annotator spent ≈40 hours reading the IEEE 29148 standard (International Organization for Standardization 2011) and the relevant chapters of two requirements engineering textbooks (van Lamsweerde 2009; Pohl 2010). Our third-party annotator annotated the 18 RSs over the course of a 10-week, full-time internship. She needed six full working days, as mentioned above, for training and reading the references on requirements. The remaining 44 days were spent on the annotation task. The 18 RSs contained between one and 270 pages (40 pages on average), corresponding to ≈11000 requirement candidates in total.

In the next step and using the NLP Sentence Splitter module discussed in Section 2, we transformed each RS into a set of sentences. We recall from Section 3 that these sentences, referred to as requirement candidates, are the units for classification. Applying sentence splitting to the 30 RSs resulted in a total of 18306 requirement candidates. We mapped these requirement candidates onto the manually identified requirements regions as per the annotations provided by our industry partner and third-party annotator. Specifically, any requirement candidate whose span intersected with a (manually specified) requirements region was deemed as a requirement. All other requirement candidates were deemed as non-requirements. This process led to 7351 requirement candidates marked as requirements and 10955 requirement candidates marked as non-requirements.

As a quality measure and since our third-party annotator was not a software specialist, two researchers (authors) independently verified her work. In particular, the two researchers examined a random subset of the content pages of the 18 RSs processed by the annotator and marked the requirements in these pages. This subset, which was divided almost equally between the two researchers, accounted for ≈20% of the content of the underlying RSs and contained ≈2200 requirement candidates. The interrater agreement between the annotator and the two researchers was computed using Cohen's Kappa (κ) (Cohen 1960). A requirement candidate counted as an agreement if it received the same classification by the annotator and a researcher, and as a disagreement otherwise. The obtained κ scores were, respectively for the two researchers: 0.87 ("strong agreement") and 0.91 ("almost perfect agreement") (McHugh 2012). The disagreements were almost exclusively over requirement candidates that, due to the annotator's lack of domain expertise, could not be classified with adequate confidence. Since no notable anomalies were detected during verification, the annotator's results were accepted without modification.

We partition the requirement candidates into a training set, T, and a validation set, E. We use T for training a classification model and E for evaluating the trained model. Following standard best practices, we set the split ratio between T and E to 9:1, subject to the condition that *no* RS should contribute requirement candidates to both T and E. This condition is essential for ensuring the realism of our validation: in practice, our approach will be applied to RS no parts of which have been exposed to the approach during training. Due to the above condition, a sharp 9:1 split could not be achieved; our goal was thus getting

the number of requirement candidates in E to be as close as possible to 10% of the total number of requirement candidates. To further increase the validity and generalizability of our results, we enforced the following additional criteria: (1) The RSs in E must all have distinct origins (sources) and distinct application domains, (2) The RSs in E must have different origins than those in T. Our split strategy resulted in an E with four RSs, RS1 to RS4, as described in Table 3. We refer back to these RSs when answering RQ3. The four RSs contain 2145 requirement candidates, of which 663 are requirements and 1482 are non-requirements. These RSs constitute 11.7% of the requirement candidates in our dataset (9% of all requirements and 13.5% of all non-requirements).

5.2.2 Expert Interview Survey

We conducted three interview surveys with experts to assess their perception of the usefulness of our approach in practice. The survey material consist of three RSs (RS5–RS7), automatically demarcated by our tool as described in Section 3. Table 4 shows an overview of RS5–RS7. For each RS, the table reports the total number of pages in the RS, the number of relevant (demarcated) pages in the RS (explained later), and the number of requirement candidates in the RS marked as "Requirement" or "Non-requirement" by our approach.

For RS5 and RS6, the interview surveys were conducted with an expert from our industrial partner in the aerospace domain. For RS7, three experts from our collaborating partner, QRA Corp, participated in the interview survey. The four experts across the three interview surveys have a collective experience of more than 30 years in requirements engineering and specification.

Using Likert scales, our survey was designed to elicit information from experts over four statements (Statement 1–4) and two follow-up statements (Statement 1-F and Statement 2-F) listed in Fig. 5. The experts were asked to rate the Statements 1–4 on each page in the RS. The experts' feedback for these statements was solicited on a per-page basis to have a consistent unit of analysis, irrespective of the number of candidates demarcated on a given page. Statement 1-F and 2-F were asked as follow-up to Statements 1 and 2, depending on the experts' answers. We note that the phrase "automated support" in the questionnaire refers to our demarcation approach.

Statement 1 concerns *false negatives* or missed requirements. For each missed requirement, the experts answered a follow-up statement: Statement 1-F. In most requirements automation tasks, such as requirements demarcation which is the focus of our work, finding a missed correct answer can be difficult (Berry 2017). Statement 1-F captures the experts' perception of the ease of identifying missed requirements, given the cues provided by our

Table 3 The RSs in our validation Set (E)

Id	Description of the RS	Number of requirements	Number of non-requirements
RS1	Requirements for a sensor-based industrial control system	112	250
RS2	Requirements for standardizing the structure of defense contracts	139	348
RS3	Requirements for an astronomical observatory	173	509
RS4	Requirements for a space exploration vehicle (rover)	239	375

Table 4 The RSs evaluated in our expert interview surveys

Id	RS5	RS6	RS7
RS Description	Requirements for a deep space spacecraft mission	Requirements for an air traffic management system	Requirements for a library resource management system
Total pages	45	51	20
Demarcated pages	37	46	13
Candidates marked by the tool as:			
Requirement	281	294	113
Non-requirement	674	818	226

approach. For example, had our approach missed the demarcation of R6 as a requirement in Fig. 1, it would have been relatively easy for an expert to spot the missing demarcation, given the cues conveyed by the surrounding demarcations.

Statement 2 concerns *false positives*, i.e., demarcated non-requirement candidates. For each such candidate, the experts answered Statement 2-F, which informs us about whether the false positive provides any useful information about the system under consideration. For example, in Fig. 1, both N1 and N3 are non-requirements, but N3 certainly provides more important information about the system than N1, and is likely to trigger discussions among stakeholders during requirements quality assurance activities, such as reviews and inspections.

Statement 3 gathers the experts' perception about the efficiency of requirements demarcation when using automated support. Statement 4 addresses the likelihood of experts missing out on a requirement or important information when working without automated support. As discussed in Section 1, requirements might be unintentionally defined in parts of an RS where experts do not expect to find requirements, e.g., R5 in Fig. 1 is defined within the supplementary information segment of R4. Statement 4 is targeted at examining such cases, where experts could potentially overlook – when working within their usual time budget and without automated assistance – requirements or otherwise important information.

Statement 1. On this page, indicate all requirements which have not been demarcated by the automated support.
Statement 1-F. (Asked for each missed requirement) The cues conveyed by the surrounding demarcations from the automated support led me to easily spot the missed requirement.

❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree

Statement 2. On this page, indicate all demarcated candidates which are not requirements.
Statement 2-F. (Asked for each candidate deemed non-requirement) The demarcated candidate is not a requirement, but provides useful information that would warrant further discussion.

❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree

Statement 3. On this page, I would do the demarcation faster with the cues conveyed by the surrounding demarcations from the automated support than without the automated support.

❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree ❑ Not Relevant

Statement 4. On this page, given my time budget in daily practice, it is likely that I would have missed some important information if I had done the demarcation manually.

❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree ❑ Not Relevant

Fig. 5 Expert Interview Survey Questionnaire

Each survey was conducted in a single session. Each session was restricted to a maximum duration of two hours to avoid interviewee fatigue. Given the limited availability of experts and time restrictions, all experts were provided with the demarcated RS at least two days in advance and were asked to familiarize themselves with the demarcated RS content. At the beginning of each session, we explained all the statements in Fig. 5 to the experts, along with examples. For Statements 1 and 2, experts answered by identifying *false negatives* and *false positives*, respectively. For Statements 1-F, 2-F, 3 and 4, the experts rated the statements on a five-point Likert scale (Likert 1932). For those pages of the RS which clearly did not contain requirements, such as table of contents and glossary pages, the experts were provided with the additional option of choosing "Not Relevant" for Statements 3 and 4. The results from these pages have been excluded from our analysis in RQ6 as well as from the second column of Table 4.

To ensure that experts had a clear understanding of each statement, they were asked to verbalize their rationale for the first five pages they reviewed. In the interview survey of RS7, the experts further explained their rationale whenever they disagreed among themselves. The results of our interview surveys are explained in the discussion of RQ6.

We note that our expert interview surveys are exploratory in nature. Ideally, for the interview surveys to be considered as confirmatory, one would need to interview many experts from diverse domains. In our research, however, conducting interview surveys at a large scale was infeasible, noting the need for participants who are experts not only in requirements engineering but also in the subject matter of specific RSs. We performed our interview surveys with experts who individually had at least five years of experience in requirements engineering and who, in addition, had considerable familiarity with the underlying RSs. To mitigate risks related to expert errors and to increase confidence in our survey results, our interview surveys cover RSs that have a reasonably large number of requirement candidates (at least 20 pages and more than 100 requirement candidates, as shown in Table 4). In other words, each expert reviewed the performance of our approach on a large number of requirement candidates.

5.3 Characteristics of Collected Data

Training Set. We created our training set from 26 RSs varying in size from one page to 270 pages, with a total of 16161 requirement candidates. The domains covered in the training set are diverse and include information systems, automotive, healthcare, aviation, aerospace, maritime, telecommunications, control systems, and networks. Eight RSs provide no ids to distinguish requirements from non-requirements. Since we use proprietary documents, we provide more details about the distribution of the most influential features in the following.

- *hasMFModalVerb:* 5239 candidates contain the most frequently used modal verb in the RSs, of which 120 are non-requirements, while 10922 do not contain the most frequent modal verb, of which 1569 are requirements.
- *hasModalVerb:* 6798 candidates contain modal verbs, of which 913 are non-requirements, while 9363 candidates do not contain modal verbs, of which 803 are requirements.
- *hasNPModalVP:* 6736 candidates contain the sequence "a noun phrase followed by a modal verb followed by a verb phrase", of which 900 are non-requirements; whereas 9425 candidates do not contain the sequence, of which 852 are requirements.

- *numAlphabetics:* this is a numeric feature, with minimum, maximum, and mean values of 0, 92, 14, respectively. Note that a part of the table of content or a part of the RS title might contain no alphabetical words.
- *numTokens:* this is another numeric feature with minimum, maximum and mean values of 0, 163 and 17. Note that a requirement candidate which contains an equation can easily have $>$ 100 tokens.

Test Set. The four RSs we used for evaluation differ in domain (as shown by Table 3) and structure as we discuss next. **RS1** contains 19 pages in which the requirements are preceded with id patterns in the form of "Cdd" (see Section 3 for more details about how we derive alphanumeric patterns). There are a lot of conditional (if-)statements and no list environments within the requirements or the non-requirements. The percentage of requirements containing (different) modal verbs is 61%. We note that RS1 is a special case where both requirements and non-requirements are well-structured. In RS1, the non-requirements that are outside the requirements section are not prefixed with structural elements like IDs, not even for the section headers. Within the requirements section, all non-requirements are preceded with trigger words (e.g., "Rationale"). These characteristics have a significant impact on demarcation accuracy for RS1, as shown in Section 5. **RS2** contains 25 pages where requirements (as well as non-requirements) can be preceded with id patterns in the form of "d.d(.d)" (the third digit for hierarchical requirements). This id pattern is the same as the one used for section headers and other structural elements in the RS; in other words, ids can be prefixes for non-requirements too. There are few list environments starting with (c) ids. The percentage of requirements with modal verbs is 60%. **RS3** has 39 pages, 12 of which provide supplementary information. The requirements are structured using ids of the form "CC-CCC-ddd". The percentage of requirements that contain modal verbs is 65%. Out of the total number of requirements, 8% contain list environments. Finally, **RS4** contains 26 pages where the requirements are organized using id patterns of the form "[CC-CCC-CCC-ddd]". The percentage of requirements with modal verbs is 62%. The document contains about 10% of its requirements within list environments, some of which are nested lists. List environments are also used within non-requirements.

Expert Survey Documents. The three RSs used in our expert interview surveys are from two different domains: RS5 and RS6 are from the *aerospace* domain and RS7 is from the *information systems* domain, as shown in Table 4. We had the following criteria in mind while selecting these RSs: (i) we wanted to evaluate our approach on RSs for which we would have direct access to the domain experts for conducting our surveys; (ii) we were interested in RSs that were reasonably large ($>$ 10 pages), but not too large. Our approach needs to be executed on reasonably large documents to be deemed beneficial in practice by experts. However, we also had to take into consideration the time required by experts to analyze an RS and answer our questionnaire during the survey; and (iii) we wanted to cover RSs with distinct structures. All three RSs vary significantly in terms of the use of identifiers in requirements, use of lists, and supplementary information in the documents, e.g., introduction and project scope information.

5.4 Metrics for the Evaluation of ML Classification Models

We use standard metrics, *Accuracy (A)*, *Precision (P)* and *Recall (R)* (Witten et al. 2016), for evaluating ML classification models. *Accuracy* is computed as the ratio of requirement candidates correctly demarcated as requirement and non-requirement to the total number of

requirement candidates. *Precision* is the ratio of requirement candidates correctly classified as requirement to all requirement candidates classified as requirement. *Recall* is the fraction of all requirements correctly demarcated. In our context, we seek very high recall to minimize the risk of missed requirements and acceptable precision to ensure that analysts would not be overwhelmed with false positives.

Formulas used in this evaluation. We define *TP* and *TN* are the number of true positives and true negatives, i.e., correctly classified instances of requirements and non-requirements. FP is the number of false positives, i.e., non-requirements predicted as requirements, and FN is the number of false negatives, i.e., the requirements classified as non-requirements.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{1}$$

$$\text{Precision} = \frac{TP}{(TP + FP)} \tag{2}$$

$$\text{Recall} = \frac{TP}{(TP + FN)} \tag{3}$$

5.5 Evaluation Procedures

The procedures used for answering the RQs are as follows:

Algorithm Selection. ML classification algorithms can be broadly categorized into mathematical, hierarchical, and layered algorithms (Suthaharan 2016). In our evaluation, we considered, from each of these three categories, widely used and recommended algorithms (Louridas and Ebert 2016). Specifically, we consider and compare five ML algorithms: *Logistic Regression* and *Support Vector Machine* (mathematical category), *Decision Tree* and *Random Forest* (hierarchical category), and *Feedforward Neural Network* (layered category). All algorithms are tuned with optimal hyperparameters that maximize classification accuracy over T. For tuning, we apply multisearch hyperparameter optimization using random search (Reutemann et al. 2018; Bergstra and Bengio 2012). The basis for tuning and comparing the algorithms is ten-fold cross validation on T. In ten-fold cross validation, a given dataset is randomly split into ten equal subsets. Nine subsets are used for training and the last subset for evaluation. The procedure is repeated ten times for predicting on all subsets. The ML algorithm that yields the best average accuracy across the ten folds is selected.

We further examine the usefulness of cost-sensitive learning, discussed in Section 2.1. To do so, we run the ten-fold cross validation procedure above both with and without cost-sensitive learning. For cost-sensitive learning, we assign false negatives double the cost (penalty) of false positives. The algorithm selection procedure is used for answering RQ1.

Feature Importance Analysis. We assess the importance of the features of Table 1 using information gain (IG) (Witten et al. 2016) computed on T. Intuitively, IG measures, in our case, how efficient a given feature is in discriminating requirements from non-requirements. A higher IG value signifies a higher discriminative power. IG is used for answering RQ2.

Model Validation. We evaluate the best-performing ML algorithm by training it on the RSs in T and applying the resulting classification model for demarcating the RSs in E. This procedure is used for answering RQ3.

Comparison with Baselines. Our approach is useful only if it outperforms simple automated solutions that are based on writing and markup conventions. To this end, we compare against five baseline solutions. These are: **(B1)** marking as requirement any requirement candidate containing a modal verb; **(B2)** marking as requirement any requirement candidate containing the most frequent modal verb of the underlying RS; **(B3)** marking as requirement any requirement candidate beginning with an alphanumeric pattern; **(B4)** taking the union of the results from B1 and B3; and **(B5)** taking the union of the results from B2 and B3. Our comparison with these baselines is performed over E and discussed as part of RQ3.

Tradeoff Analysis. As stated in Section 5.1 (RQ5), we are interested in assessing the benefits of our features against the execution time incurred by them. Noting that NLP dominates the execution time of our approach, we examine alternative ways of simplifying our NLP pipeline. Naturally, the exclusion of any NLP module comes at the expense of some features no longer being computable. The question is whether the quality degradation that results from not using certain features is acceptable. Since the alternatives to consider are few, we investigate the tradeoffs through exhaustive analysis. Specifically, we group our features based on their prerequisite NLP modules and compute the classification evaluation metrics for different combinations of feature groups. We then determine whether any of the combinations leads to tangible reductions in execution time without compromising classification quality. This tradeoff analysis, which is meant at answering RQ5, is done via ten-fold cross validation over our entire dataset ($T \cup E$).

5.6 Discussion

Below, we answer the RQs posed in Section 5.1.

RQ1. Table 5 shows the accuracy, precision, and recall results for the five ML classification algorithms considered. These results were computed on the training set (T) through ten-fold cross validation, both with and without cost-sensitive learning, denoted CSL and ¬CSL, respectively. All algorithms had tuned hyperparameters. To improve readability, in this and all the subsequent tables that report on classification quality, we highlight in **bold** the best accuracy, precision, and recall results. From Table 5, we conclude that **Random Forest presents an advantage over the alternatives** across all three metrics. This is further confirmed by the hypothesis test below. We answer the remaining RQs using Random Forest as our classification algorithm.

Hypothesis Test. A common statistical test used for checking the difference between two proportions is the z-test (Dietterich 1998). In our case, we conduct z-tests to compare the error rates of each pair of algorithms–denoted as X_A and X_B with error rates p_A and p_B below. The error rate is the proportion of requirement candidates misclassified by an algorithm. This means that, if algorithm X_A is applied on a set with n requirement candidates, then p_A is computed as the number of requirement candidates misclassified by X_A (i.e., requirements that are wrongly classified by X_A as non-requirements, and vice versa) divided by n. To conduct the z-tests, we first train the five classifiers on 90% of our training set; the selection of this 90% from the entire training set is random. We then apply the resulting classifiers to the remaining 10% of the data (comprised of

Table 5 ML algorithm selection results (RQ1)

		¬CSL	CSL
Feedforward Neural Network	Accuracy(%)	88.9	85.6
	Precision(%)	84.3	76.5
	Recall(%)	90.0	94.3
Decision Tree	Accuracy(%)	92.9	92.1
	Precision(%)	91.1	86.9
	Recall(%)	91.7	95.4
Logistic Regression	Accuracy(%)	92.8	92.5
	Precision(%)	92.1	87.3
	Recall(%)	90.4	95.7
Random Forest	Accuracy(%)	**94.4**	93.7
	Precision(%)	**94.3**	88.9
	Recall(%)	91.9	**96.9**
Support Vector Machine	Accuracy(%)	92.1	91.4
	Precision(%)	90.1	87.5
	Recall(%)	91.0	92.4

1616 requirement candidates). The hypotheses that we want to test are:

$$H_0 : p_A = p_B \text{ (null hypothesis)}$$
$$H_1 : p_A \neq p_B$$

For comparisons with Random Forest, the z-scores and p-values resulting from our tests are summarized in Table 6. The results indicate that, at the 0.05 significance level, the null hypothesis *is rejected* in all comparisons. Moreover, we performed a one-way ANOVA test on the accuracy, precision and recall results across the ten-folds involved in cross validation and for all classification algorithms. The null hypothesis is that the means of the different algorithms for the different metrics are equal. This additional test results in a *p-value<0.001* for all considered metrics, and hence supports the rejection of the null hypothesis. We conclude that, for requirements demarcation, empirical evidence suggests that the error rate of Random Forest is (statistically) significantly less than that of the other algorithms considered in our analysis.

Table 6 Z-test results for random forest

Test	p_A	p_B	Mis-classified by X_A	Mis-classified by X_B	z-score	p-value
1	Random forest	Feedforward neural network	87	232	−8.551	0
2	Random forest	Decision tree	87	117	−2.17	0.003
3	Random forest	Logistic regression	87	114	−1.967	0.0492
4	Random forest	Support vector machine	87	135	−3.338	0.0008

RQ2. Fig. 6 lists the features of Table 1 in descending order of information gain (IG). Based on the results in this figure, **the most influential features are hasMFModalVerb, hasModalVerb, hasNPModalVP, hasVerb, numAlphabetics, and numTokens.** The top-three features – *hasMFModalVerb*, *hasModalVerb*, and *hasNPModalVP* – all have to do with modal verbs.

The high IG scores of these three features is a clear indication that taking note of the presence or absence of modal verbs is essential for telling requirements apart from non-requirements. The next group of important features – *hasVerb*, *numAlphabetics*, and *numTokens* – are targeted at excluding non-requirements. With the exception of *startsWithId* and *startsWithDetVerb*, all the remaining features turn out to be useful, albeit to a lesser extent than the most important features discussed above. Nevertheless, when considered collectively, these less important features still have considerable influence on classification.

The IG score of zero obtained for *startsWithId* indicates that the mere presence of (alphanumeric) identifiers is not a useful factor for classifying requirements and non-requirements. This observation can be explained by the fact that many non-requirements, e.g., section headers, may be numbered too. While identifiers per se are not useful for requirements demarcation, the aggregation of identifier information with frequencies turns out to be important, as indicated by the IG score of *idPatternFrequency*. The second and last feature with an IG score of zero is *startsWithDetVerb*, indicating that the linguistic pattern represented by this feature does not contribute to distinguishing requirements from non-requirements.

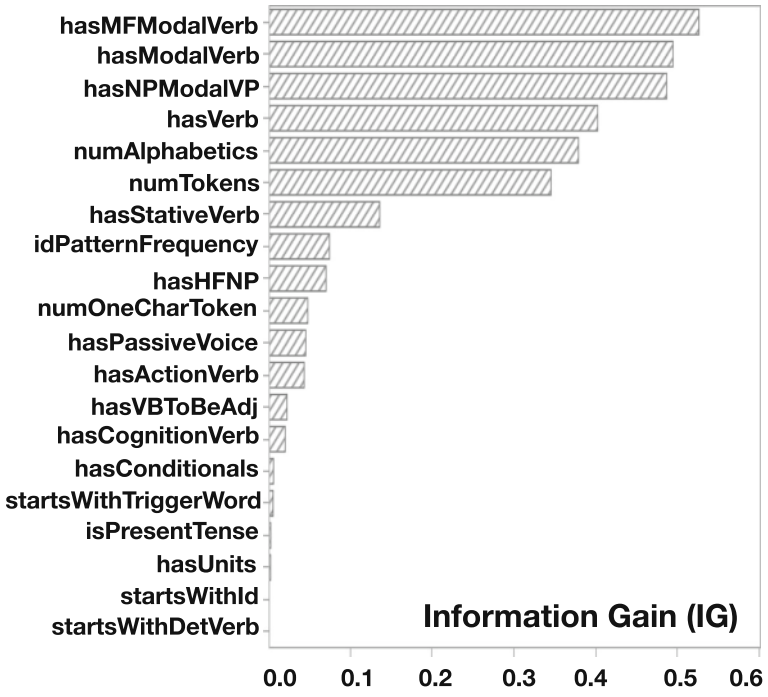


Fig. 6 Feature Importance (RQ2)

In summary, the results of Fig. 6 provide evidence that all but two of our features (*startsWithId* and *startsWithDetVerb*) indeed play a role in requirements demarcation, thus confirming the majority of the intuitions presented in Table 1.

RQ3. Table 7 shows the accuracy, precision, and recall results obtained by training Random Forest (RF) – the best classification algorithm from RQ1 – over the training set (T), and then applying the learned classification model to the validation set (E), i.e., RS1–RS4 in Table 3. Like in RQ1, we trained the model both with and without cost-sensitive learning (CSL and ¬CSL). To enable comparison, the table further shows the results from applying to RS1–RS4 the five baseline solutions discussed in Section 5.5 (*Comparison with Baselines*).

We observe from Table 7 that our classification model (1) outperforms all the baselines in terms of accuracy, irrespective of whether cost-sensitive learning is used or not, (2) outperforms all the baselines in terms of recall, when trained with cost-sensitive learning. As for precision, **B2** performs better than our classification model on three of the RSs (RS2, RS3, and RS4). However, **B2** has a considerably lower recall: on average, compared to the classification model with cost-sensitive learning, **B2** has 13.4% better precision but 45.7% worse recall; and compared to the model without cost-sensitive learning, **B2** is better by 6.5% in terms of precision but worse by 36.1% in terms of recall. Similarly, **B1** has better precision on one of the RSs (RS2) than our classification model with cost-sensitive learning (difference of 10.1%); nevertheless, **B1**'s recall is much lower (difference of 32.3%).

Table 7 Model validation results (RQ3)

		RS1	RS2	RS3	RS4
RF (¬CSL)	Accuracy(%)	95.6	**92.4**	**92.1**	87.5
	Precision(%)	**100**	85.4	83.5	84.6
	Recall(%)	85.7	88.5	87.3	82.2
RF (CSL)	Accuracy(%)	**99.7**	88.1	89.6	**88.3**
	Precision(%)	**100**	71.2	72.8	80.6
	Recall(%)	**99.1**	**97.8**	**94.2**	**91.6**
B1	Accuracy(%)	63.8	85.8	84.3	67.3
	Precision(%)	44.0	81.3	70.6	57.3
	Recall(%)	62.5	65.5	65.3	62.3
B2	Accuracy(%)	81.2	84.4	87.1	82.4
	Precision(%)	92.3	**97.0**	**91.3**	**97.8**
	Recall(%)	42.9	46.8	54.3	56.0
B3	Accuracy(%)	82.0	52.8	81.8	79.2
	Precision(%)	66.9	29.0	62.1	71.8
	Recall(%)	83.0	45.3	72.8	76.6
B4	Accuracy(%)	58.6	61.6	79.8	64.7
	Precision(%)	41.8	41.7	56.5	52.9
	Recall(%)	86.6	86.3	88.4	84.1
B5	Accuracy(%)	80.9	61.0	83.9	81.1
	Precision(%)	65.0	40.2	63.5	72.9
	Recall(%)	83.0	74.8	85.5	82.0

In light of the above and given that, in our context, recall takes precedence over precision as argued before, we can conclude the following: **None of the baseline solutions provide a compelling alternative to our ML-based approach.** At the same time, we note that our better results are by no means a refutation of the common-sense intuition behind the baselines. Indeed, our approach incorporates the *baselines through closely related features. In particular, hasModalVerb* and *hasNPModalVP* relate to **B1**, *hasMFModalVerb* to **B2**, and *startsWithId* and *idPatternFrequency* to **B3**. As we discussed in RQ2, all our features except *startsWithId* and *startsWithDetVerb* are relevant. The main observation from our analysis is therefore that **no individual baseline or combination of baselines is adequate without considering in tandem the more nuanced characteristics of the content in RS.**

As noted in Section 5.5, we use cost-sensitive learning for giving more weight to recall than precision. **Our approach without cost-sensitive learning has an average precision of 88.1% and average recall of 86.1% over RS1–RS4. Cost-sensitive learning increases recall to 95.7% (gain of 9.6%) while decreasing precision to 81.2% (loss of 6.9%).** In absolute terms, this amounts to trading 59 fewer missed requirements (false negatives) for 80 non-requirements misclassified as requirements (false positives). While the impact of cost-sensitive learning on manual effort is difficult to quantify without a user study and is not addressed in this article, **the engineers at our industry partner favored using cost-sensitive learning**: they perceived the effort of filtering the additional false positives to be a reasonable price to pay for missing less requirements. **In comparison to the five baseline solutions, our approach with cost-sensitive learning has, on average, 16.4% better precision and 25.5% better recall.**

When employed with cost-sensitive learning, step 3 of our approach (Fig. 3) misclassifies as non-requirement a total of 34 requirements across RS1–RS4. The post-processing (step 4 in our approach) of the classification results reduces the number of false negatives to **21**. In our experiments, the post-processing step introduced no further false positives. Although there is no guarantee in general that the number of false positives will not be increased by post-processing, our experiments support the hypothesis that the list environment appears less often within non-requirements. Table 8 shows the results after performing post-processing on our test set.

We analyzed all the misclassifications in order to determine their root causes. We identified three root causes, as shown in Table 9. Each row in the table explains one of the causes, illustrates it with an example, and reports the number of misclassifications attributable to that cause.

For the seven cases in row 1, our approach demarcated only one part of the requirement that spans over multiple sentences. In practice, when an analyst reviews the automatically generated demarcations, these incomplete cases are relatively easy to spot and fix, as witnessed in our interview surveys, discussed later in RQ6.

Table 8 Post-processing results (RQ3)

		RS1	RS2	RS3	RS4
RF (CSL)	Accuracy(%)	99.7	88.3	90.0	89.6
	Precision(%)	100	71.2	72.8	80.6
	Recall(%)	99.1	98.6	95.9	95.2

Table 9 Root causes for false negatives

Cause	Explanation	Count
1) Loss of context	The units of analysis in our approach are sentences. A requirement that is specified over multiple sentences would constitute multiple classification units. Sometimes, one unit within such requirements is correctly classified whereas the remaining unit(s) might not be, resulting in misclassification. **Example:** In the requirement below, we have two sentences (requirement candidates). candidate$_1$ The control interface shall be tolerant & function, if the round-trip latency < 250ms. The interface further needs to be tolerant to the jitter conditions specified in AD1-OPC11. candidate$_2$ We correctly demarcate candidate$_1$ as requirement, but miss candidate$_2$ (one false negative).	7
2) NLP errors	NLP techniques are not fully accurate and make mistakes. This is particularly true when these techniques are confronted with statements that significantly differ from normal text, e.g., statements with numerous abbreviations or statements beginning with complex labels. NLP errors may lead to incorrect feature extraction and thus classification errors. **Example:** we miss the following requirement because our NLP pipeline does not process it correctly. *"3.2.2.4.1 Default Ramp Angle - The default ramp andle (REF27 in SD 119) for the controller should, upon ACK, be set to 34°."*	8
3) Conservative ground truth	As noted in Section 5.2.1, our third-party annotator was instructed to favor the requirement class when in doubt. In our error analysis, we observed borderline situations where automated classification could well be correct, but did not match the deliberately conservative ground truth. **Example:** The following sentence is a requirement in our ground truth but is classified as non-requirement by our approach. *"The decision to retain any single=point failures of any severity level in the design is subkect to formal approavals on a case-by-case basis, with a detailed analysis for each failure."*	6

Nonetheless, we elected to count these cases as false negatives. The eight cases in row 2 are unavoidable due to NLP seldom being fully accurate. The six cases in row 3 are marginal situations where we could not decide whether the automated classification was at fault or the manual annotations (ground truth) were overly conservative. Again, we found it more sensible to treat these cases as false negatives.

RQ4. We answer RQ4 using a computer with a 3GHz dual-core processor and 16GB of memory. We consider the execution time of our approach both from a solution provider's perspective and from a user's perspective. Both the provider and the user need to run the first two phases of our approach, namely, document parsing and feature matrix construction as explained in Section 3. In the case of the provider,

these two phases will be executed on a large corpus of *already annotated* RSs for the purpose of model training. In the case of the user, the two phases are applied over an individual RS in preparation for the third phase of the approach, namely classification.

Over our training set (T) which is composed of 16161 requirement candidates, the first two phases of the approach took 27.3 minutes to run, i.e., an average of ≈101 milliseconds per requirement candidate. Training the classification model took negligible time (16 seconds). The training execution time does not include the time required to curate and annotate our training documents. As discussed in Section 5.2.1, our third-party annotator spent a total of 352 hours on annotating our training RSs, equivalent to an average annotation time of 1.92 minutes per requirement candidate. **The training execution and annotation time is acceptable from the provider's standpoint, since training is a one-off and performed only occasionally as the training set is revised or expanded.**

From the user's standpoint, performing the first two phases of our approach over the validation set (E) led to the following results: 37 seconds for RS1, 50 seconds for RS2, 72 seconds for RS3, and 67 seconds for RS4; these RSs collectively contain 2145 requirement candidates giving an average processing time of ≈105 milliseconds per requirement candidate. The time required for the third phase, i.e., classification, was negligible (<1 second per RS). Based on these results, if we assume an average of 30 requirement candidates per page in an RS, the end user should anticipate ≈3 seconds of processing time per page. **Such an execution time is adequate for batch (offline) processing on an RS.** In RQ5, we attempt to optimize the execution time in order to make our approach more suitable for interactive analysis.

Complexity. We discuss the (theoretical) computational complexity of our approach based on *Stanford CoreNLP* (Manning et al. 2014) – one of the most common NLP toolkits covering the NLP pipeline required by our approach (Hirschberg and Manning 2015) – and the Random Forest implementation in WEKA (Breiman 2001). Let n be the number of sentences in the input RS and m be the length of a sentence. Then, tokenization, sentence splitting and POS-tagging, which are the first steps in the NLP pipeline (see Fig. 2), require linear time with respect to the sentence length, i.e, have a complexity of $\mathcal{O}(n * m)$ (Manning et al. 2014). Constituency parser requires cubic time with respect to the sentence length, i.e., a complexity of $\mathcal{O}(n * m^3)$, while the dependency parser can be much faster with a complexity of $\mathcal{O}(n * m^2)$ (Covington 2001). Semantic parsing is limited to looking up, for a given sentence, the semantic categories of the verbs as given by WordNet. The complexity of this implementation is most often discussed based on the time needed to get a response on a query from WordNet (Geller et al. 1997). In our experiments, this response time equals to *one millisecond per requirement candidate*. Training a Random Forest (RF) model has a time complexity of $\mathcal{O}(r * d * n^2 * \log n)$ whereas classifying using a pre-trained RF model requires $\mathcal{O}(r * d)$, where r is the number of randomized trees, d is the maximum depth of the tree and n is the number of training examples (Louppe 2014).

The analysis performed above indicates that there are two dominant complexity factors: constituency parsing and training the RF model on a large number of

requirement candidates. First, we note that m – the cubic term in the time complexity of constituency parsing – tends to be small in our document collection and can thus be regarded as a constant. This is in fact true for the vast majority of NL documents, noting that arbitrarily long sentences are virtually absent. This said, constituency parsing is still expensive since m^3 would still be a large coefficient. Therefore, in *RQ5*, we analyze through experimentation the effect of excluding constituency parsing from the NLP pipeline and show that such an exclusion would not drastically affect accuracy. Second, and with regard to training the RF model, we note that this training needs to be performed only occasionally whenever the training set is updated. Since this can be done offline (e.g., overnight), we believe that the computation time can be afforded. Therefore, overall, we do not anticipate difficulties in terms of computational complexity over RSs.

RQ5. To compute the token-based features of Table 1, one needs to execute only the preprocessing portion of the NLP pipeline in Fig. 2. The syntactic and frequency-based features additionally require constituency and dependency parsing; whereas the semantic features additionally require semantic parsing (but not constituency or dependency parsing). These prerequisite relations induce four groups of features: (1) only token-based features, denoted *Tok*, (2) the combination of token-based, syntactic and frequency-based features, denoted *TokSynFrq*, (3) the combination of token-based and semantic features, denoted *TokSem*, and (4) all features, denoted *All*.

In Table 10, we show for each group of features the results of ten-fold cross validation over our entire dataset alongside the time it took to run the prerequisite NLP modules and compute the features in that group. Following the conclusions from RQ1 and RQ3, we use Random Forest with cost-sensitive learning for classification. The execution times reported in the table are averages per requirement candidate and given in milliseconds. We observe that *Tok* and *TokSem* are inexpensive to compute and achieve good recall. However, these two feature groups lead to drastically lower precision – by a factor of 20% – than *TokSynFrq* and *All*. At the same time, *TokSynFrq* is not a better alternative than *All* either, since it slightly reduces classification quality while offering no tangible speedup.

The fact that syntactic and frequency-based features explain most of the execution time prompted a followup investigation. In particular, we looked into whether the exclusion of any of these features would allow us to make the NLP pipeline more efficient, without significantly impacting classification quality. We observed that there is only one feature, Syn5, requiring a constituency parse tree. To compute the remaining syntactic features and the frequency-based features that rely on syntactic analysis (*hasMFModalVerb* and *hasHFNP*), one can replace

Table 10 Demarcation quality vs. execution time (RQ5)

		Tok (1.2 ms/c)	TokSynFrq (101 ms/c)	TokSem (2 ms/c)	All (102 ms/c)	All-{Syn5} (49 ms/c)[1]
RF (CSL)	Accuracy(%)	79.9	93.5	80.9	**93.8**	93.6
	Precision(%)	67.6	88.4	68.8	**88.9**	88.6
	Recall(%)	95.7	96.4	96.0	**96.5**	96.4

[1]Best Tradeoff

Table 11 Results of interview surveys

	RS5	RS6	RS7
Demarcated pages	37	46	13
Candidates marked as requirement	281	394	113
True positives	213	359	93
False negatives†	1	8	1
False positives‡	68	35	20
Precision(%)	75.8	91.1	82.3
Recall (%)	99.5	97.8	98.9

†Results of **Statement 1**. On this page, indicate all requirements which have not been demarcated by the automated support

‡Results of **Statement 2**. On this page, indicate all demarcated candidates which are not requirements

constituency parsing with text chunking (shallow parsing) (Ramshaw and Marcus 1999). Text chunking has already been shown to be a robust and accurate alternative to constituency parsing for extracting the atomic-phrase structure of textual requirements (Arora et al. 2015, 2017). Based on the RQ2 results, Syn5 (*hasConditionals*) contributes very little to classification. **Excluding Syn5 and using text chunking instead of constituency parsing thus provides a good tradeoff for speedup. Using this configuration, denoted *All*−{*Syn5*} in Table 10, we reduce the execution time from 102 to 49 milliseconds per requirement candidate with negligible impact on classification quality.** We believe that the improved execution time is sufficient for an interactive mode of use, considering that, at any point in time, the user will be reviewing at most a handful of pages of an RS. Assuming 30 requirement candidates per page, our tradeoff solution reduces the execution time from 3 seconds to 1.5 seconds per page.

RQ6. Table 11 and Figs. 7 and 8 summarize the results from our three expert interview surveys, conducted by following the procedure described in Section 5.2.2. Table 11 provides overall statistics from the surveys, showing for each RS the number of demarcated pages on which the experts provided feedback, the number of requirement candidates demarcated, the number of candidates classified as requirement by experts (true positives), the number of requirements missed by our approach (false negatives), the number of candidates classified as a non-requirement by experts (false positives), and the corresponding precision and recall results.

Figures 7 and 8 depict, using barcharts, the expert feedback elicited for Statements 2-F, 3 and 4 of the survey questionnaire (Fig. 5). Each of the five bars plotted in Figs. 7, 8a, and b correspond to the participants' feedback on RS5, RS6 and RS7, respectively. We recall from Section 5.2.2 that one expert participated in the surveys for RS5 and RS6, and three experts participated in the survey for RS7. We further note that the barchart in Fig. 7 for Statement 2-F results is based on 68, 35 and 20 data points for RS5–RS7, respectively. The data points correspond to the number of false positives identified by the experts in Statement 2. The barcharts in Fig. 8 for Statements 3 and 4 are based on 37, 46 and 13 data points for RS5–RS7, respectively, and represent the number of pages in the RSs on which the experts provided feedback.

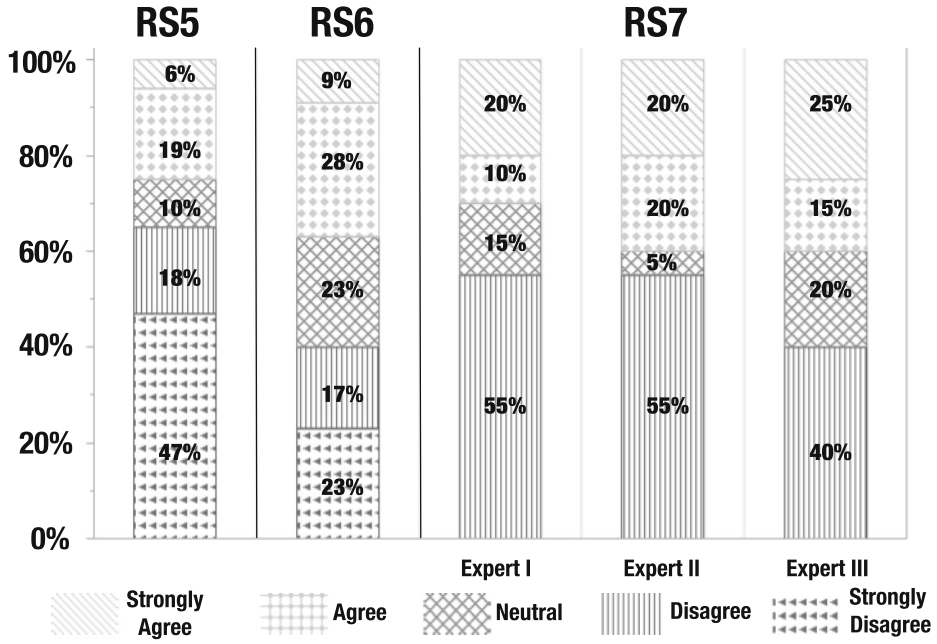
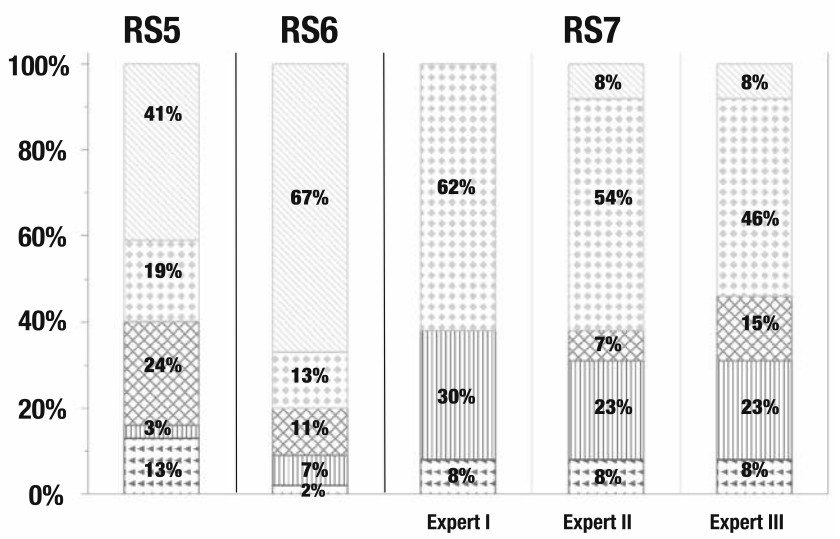


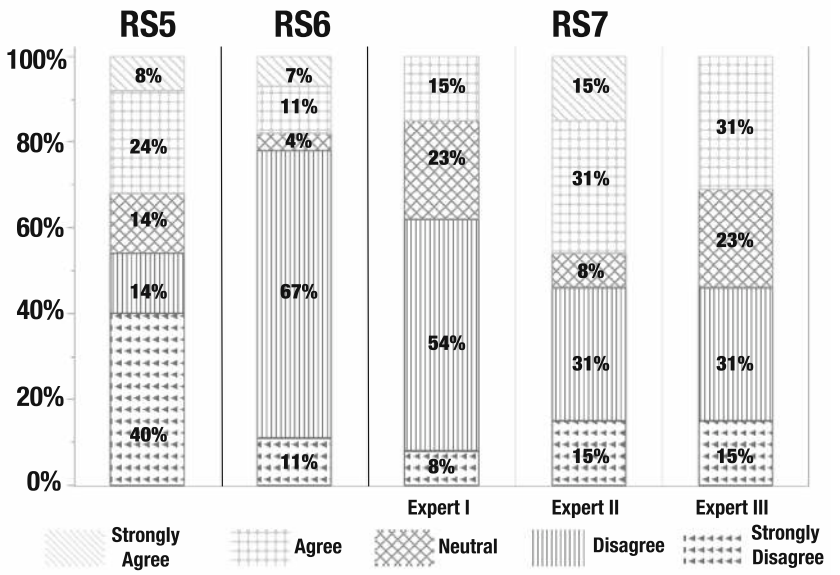
Fig. 7 Barchart for expert survey interview results of Statement 2-F

With regard to Statement 1, the experts identified a total of ten (1+8+1) false negatives in three surveys. Further, the experts classified as requirement 213 requirement candidates in RS5, 359 requirement candidates in RS6 and 93 requirement candidates in RS7, i.e., a total of 665 true positives in RS5–RS7. **The overall recall of our approach on RS5–RS7 is 98.5%.** For each of the *ten* false negatives, the experts answered the follow-up Statement 1-F (***Statement 1-F.*** *"The cues conveyed by the surrounding demarcation from the automated support led me to easily spot the missed requirement."*). Given the small number of false negatives, the response distribution for Statement 1-F is not shown in a barchart. In RS5, the expert strongly agreed that the surrounding demarcation from the automated support did help in identifying the only false negative. Similarly in RS7, all three experts responded to Statement 1-F with "Strongly Agree" for the sole missed requirement. In RS6, for eight false negatives, the expert responded once with "Agree", twice with "Neutral", and five times with "Disagree". Overall for Statement 1-F in three surveys, the experts (strongly) agreed thrice. In all three cases, the missed requirement candidate was part of a requirement specified over multiple sentences. As shown in row 1 of Table 9, our approach correctly demarcated a part of the requirement but not all the sentences in the requirement. Partially missed requirements were easy to spot for the experts. The expert in the survey for RS6 disagreed with Statement 1-F when the requirements missed were significantly different in structure from the other candidates demarcated on the page, making the missed requirements difficult to spot just through a quick glance on the page.



Statement 3. On this page, I would do the demarcation faster with the cues converted by the surrounding demarcation from the automated support than without the automated support.

(a) Statement 3 Results



Statement 4. On this page, given my time budget in daily practice, it is likely that I would have missed some important information if I had done the demarcation manually.

(b) Statement 4 Results

Fig. 8 Results for Statements 3 and 4

With regard to Statement 2, the experts classified as non-requirement a total of 123 candidates (68+35+20), i.e., false positives, in three interview surveys. **The overall precision of our approach computed over RS5–RS7 is 84.7%.** For the followup Statement 2-F, we had 163 responses in total, i.e., 68 responses for RS5, 35 responses for RS6, and 60 ($3 * 20$) responses for RS7 – as we had three expert participants. In RS5 and RS6, the expert strongly agreed or agreed in 29.1% (30/103) of the cases, was neutral in 14.6% (15/103) of the cases, and disagreed or strongly disagreed in the remaining 56.3% (58/103) of the cases that the false positives captured important information about the system under consideration. For RS7, the three experts collectively agreed (or strongly agreed) in 30% (6/20) of the cases, and collectively disagreed in 40% (8/20) of the cases for Statement 2-F. However, they had a difference of opinion over the remaining six false positives. The difference was primarily due to subjectivity about the importance of the information captured in these non-requirements. **Overall, when considering all the responses for Statement 2-F in our survey, the experts perceived more than 42% of the non-requirement candidates demarcated by our approach as containing important information, that could warrant further discussions among system stakeholders.**

With regard to Statement 3, we had 122 ($37 + 46 + 3 * 13$) responses in total. Of these, in surveys over RS5 and RS6, the expert (strongly) agreed in 71.1% (59/83) of the cases that the automated support helps improve their efficiency. The expert was neutral in 16.9% (14/83) and disagreed or strongly disagreed in 12% (10/83) of the cases. In RS7, the three experts collectively agreed or strongly agreed in 46.1% (6/13) and disagreed or strongly disagreed in 23.1% (3/13) of the cases. For the remaining four pages in RS7, the experts had a difference in opinion on whether automated support is indeed helpful in improving their efficiency for requirements demarcation on these pages. **Overall, in more than 67% of the responses for Statement 3, the experts agreed or strongly agreed that our approach helps them demarcate requirements more efficiently.** For approximately 5% of all the pages considered, the experts felt that the output from the approach is misleading and it would take them more time to demarcate the requirements with automated support than without. In a post-mortem analysis, we realized that the experts had such perception mainly for pages where all candidates demarcated on the page were classified as false positive in Statement 2, and the response for most of these false positives in Statement 2-F was "Disagree" or "Strongly Disagree".

With regard to Statement 4, similar to Statement 3, we had 122 responses in total over RS5–RS7. For this statement, we were particularly interested in positive responses, i.e., when experts (strongly) agreed that our approach helps them locate information or requirements that they might otherwise overlook. If they did not agree with Statement 4, it simply meant that they would have spotted the requirement with or without the automated support. In surveys for RS5 and RS6, out of 83 responses, the expert strongly agreed or agreed in 24% (20/83) of the cases, and was neutral in 8.4% (7/83) of the cases. In RS7, the experts collectively agreed on only one page. **Overall in three surveys, in 24.5% (30/122) of the responses, the experts agreed or strongly agreed with Statement 4.** The experts rationalized their positive responses by claiming that they did not expect to find requirements on the page and could have, in all likelihood, missed the important information had it not been demarcated by our approach.

6 Related Work

ML has been utilized as a way to provide computerized assistance for several requirements engineering tasks, e.g., trace link generation (Asuncion et al. 2010; Cleland-Huang et al. 2010; Sultanov and Hayes 2013; Guo et al. 2017; Wang et al. 2019), requirements identification and classification (Cleland-Huang et al. 2007; Winkler and Vogelsang 2016; Kurtanović and Maalej 2017a; Dalpiaz et al. 2019; Winkler et al. 2019), prioritization (Perini et al. 2013), ambiguity detection (Yang et al. 2010; Yang et al. 2012), relevance analysis (Arora et al. 2019), and review classification (Maalej et al. 2016; Kurtanovic and Maalej 2017b). The application of ML over textual requirements is almost always preceded by some form of NLP. Zhao et al. (2020) present a comprehensive overview of the applications of NLP in RE research. Among the research strands employing ML and NLP jointly for requirements analysis, our work most closely relates to the ones concerned with requirements identification and classification. Below, we compare with these strands.

Winkler and Vogelsang (2016) and Winkler and Vogelsang (2018) propose an approach based on deep learning (Goodfellow et al. 2016) for addressing the same problem that we address: requirements demarcation. They train their classifier on word embeddings (Mikolov et al. 2013) from requirements documents in the automotive domain. While we pursue the same general objective as Winkler and Vogelsang's, our solution is different in two key respects: First, Winkler and Vogelsang focus on requirements stored in IBM DOORS (2020). This enables them to narrow demarcation to distinguishing a requirement from the additional material related to that *very* requirement. In contrast, we deal with free-form RSs, meaning that we have no a-priori knowledge about the association between a requirement and its surrounding material. Second, and more importantly, Winkler and Vogelsang train their model over a specific domain (automotive), whereas our approach is domain-independent. Falkner et al. (2019) propose an ML-based approach for identifying requirements in request for proposals (RFPs) related to railway safety. They train their classifier on unique words in documents. This approach, just like Winkler and Vogelsang's, is trained on domain-specific documents. Therefore, it cannot process, due to the nature of the training data, documents from arbitrary domains the way our approach can.

There are several threads of work where ML and NLP are used together for requirements identification and classification tasks other than demarcation. Ott (2013) uses ML techniques trained on token-level information for automatically grouping requirements that belong to the same topic, e.g., temperature or voltage in automotive requirements. Cleland-Huang et al. (2007) build an iterative classifier for automated classification of non-functional requirements. The classifier learns how key indicator terms in textual requirements map onto different categories such as performance and security. Casamayor et al. (2010), Riaz et al. (2014), and Li et al. (2018) propose similar techniques based on keywords to predict categories for different requirements. Guzman et al. (2017) and Williams and Mahmoud (2017) mine requirements from twitter feeds through a combination of ML and NLP preprocessing. Rodeghero et al. (2017) use ML alongside lightweight NLP for extracting user-story information from transcripts of developer-client conversations.

The approaches discussed above are based primarily on the frequency statistics and the token/phrase-level characteristics of the underlying textual descriptions. Kurtanović and Maalej (2017a) additionally use syntactic criteria obtained from constituency and dependency parsing for distinguishing functional and non-functional requirements and

further classifying non-functional requirements into sub-categories. Dalpiaz et al. (2019) further build upon the findings of Kurtanović and Maalej (2017a) to perform functional and non-functional requirements classification using dependency parsing and lists of commonly-used verbs. Our combination of token-based, frequency-based and syntactic features as well as the use of these features in tandem with semantic ones is novel. As our empirical results in Section 5 indicate (see RQ2), all feature types are influential for an accurate differentiation of requirements and non-requirements.

7 Threats to Validity

There are two angles to the validity of our empirical work in this article: (1) the validity of the (analytical) experimentation conducted to answer RQ1–RQ5, and (2) the validity of the interview surveys undertaken to answer RQ6. The validity concerns most pertinent to the former are internal, construct and external validity; and, the validity concerns most pertinent to the latter are conclusion and external validity.

7.1 Validity of our Experimentation Study

Internal Validity. Bias was the main internal validity threat that we had to counter in our experimentation. To mitigate bias risks, the manual classification of our dataset, as we discussed in Section 5.2, was done entirely by either experts or a trained third-party (non-author). These individuals had no exposure to our demarcation tool, and were thus not influenced by its results. Until the demarcation approach was finalized and fully implemented, the researchers had no knowledge of the content of the RSs in the validation set other than the application domains and the numbers of requirements and non-requirements in these RSs; this minimal information about the validation set was necessary for planning our experimental procedures (see Section 5.5). Another potential threat to internal validity is our approach overfitting on the training data. We mitigated this threat by applying ten-fold cross validation for algorithm selection, and testing the model on RSs no parts of which had been revealed to the model during training.

Construct Validity. We treated requirements demarcation as a binary classification problem. We do not account for uncertainty, i.e., situations where a human oracle is unable to make a conclusive decision. In our evaluation, as noted in Section 5.2, we asked the annotator involved to err on the side of caution and, when in doubt, favor the requirement class over the non-requirement class. This choice is consistent with the nature of our classification problem and the need to prioritize recall over precision, as discussed in Section 2.1. Second, our units of classification are sentences. This means that we treat individual requirements spanning over multiple sentences as multiple requirements. Adapted notions of precision and recall may need to be defined, if multi-sentence requirements happen to be dominant; this was not the case in our dataset where such requirements were infrequent.

External Validity. Our experimentation was based on a relatively large dataset with the RSs in the dataset originating from a variety of sources and domains. The results obtained over our validation data is reflective of real-world conditions, particularly in that the classification model is confronted with RSs no portion of which has been revealed to the model during training. These factors combined with our consistently strong accuracy results provide confidence about the generalizability of our

approach. That said, a broader examination of requirements specification practices would be beneficial for further fine-tuning our feature set and conducting more thorough evaluations.

7.2 Validity of our Interview Survey Study

Conclusion Validity. An important threat to conclusion validity in surveys is limited survey duration. In our survey, this threat is most relevant for Statement 1 (see Fig. 5), which is rather time-consuming to assess. To mitigate this threat, we provided experts with the demarcated RSs at least two days before conducting the actual survey. The experts were asked to familiarize themselves with the demarcated content, and to identify non-demarcated information that they deemed important. Additionally and during the survey sessions, we made sure that the experts were not rushed; the experts were free to take as much time as they needed to browse through the RSs pages and identify any missed requirements.

External Validity. We conducted our interview surveys with four experts from two different domains and using three RSs with different structures. The consistency seen in the experts' feedback about the benefits of our approach is encouraging. Nonetheless, we acknowledge that the number of experts in our survey study is small. We were unable to gain access to a larger pool of experts because, as we have also observed in the past (Arora et al. 2015, 2017, 2019), senior engineers are commonly the ones who are in charge of requirements engineering activities; these engineers are small in number and further, due to their lack of availability, are difficult to solicit for time-consuming studies such as our interview surveys. Considering the small scale of our survey study, our survey results should be viewed only as suggestive and not conclusive. At the same time, we note the fact that we have been able to engage real engineers over real requirements documents, thus increasing the credibility of our survey results.

8 Conclusion

We proposed a machine learning-based approach for distinguishing requirements statements from other material in textual requirements specifications. The main characteristic of our approach is that it is applicable to a wide variety of requirements specifications without needing any input from the user. The features that we use for learning are based on linguistic and frequency information that are generalizable and meaningful irrespective of the domain and terminology of individual requirements specifications. To calculate these features for the statements in a given requirements specification, we employed a combination of natural language processing techniques. We empirically evaluated our approach using a dataset made up of 33 industrial requirements specifications. The results indicate that our approach has an average precision of 81.2% and average recall of 95.7%. We compared the effectiveness of our approach against several intuitive baselines, and demonstrated that our approach offers major benefits over these baselines. Furthermore, and using interview surveys, we collected feedback from subject-matter experts about our approach. The survey results suggest that our approach is useful in practice. The experts further found our approach to be a useful aid for identifying important information in RSs that may otherwise be overlooked. Our approach is supported by a prototype tool, named DemaRQ. This tool and the non-proprietary parts of our dataset are publicly available at https://sites.google.com/view/demarq.

Our current approach is based on binary classification. In reality, deciding between requirements and non-requirements is not always a clear-cut choice even for experts. In the future, we would like to provide more detailed information about the automatically predicted demarcations, e.g., through color coding, so that the analysts can know how conclusive the predictions are. A definitive evaluation of our approach which considers uncertainty would require user studies. Another direction for future work is to broaden the applicability of our approach beyond requirements specifications. While we do not foresee issues that would limit our current features to only requirements specifications, additional features may be necessary for accurately handling other types of requirements-relevant documents, e.g., product descriptions and calls for tenders. Further, the effectiveness of our approach over such documents needs to be evaluated via new empirical studies.

Acknowledgments This project has received funding from QRA Corp, Luxembourg's National Research Fund under the grant BRIDGES18/IS/12632261, the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), and NSERC of Canada under the Discovery, Discovery Accelerator and CRC programs.

References

Abualhaija S, Arora C, Sabetzadeh M, Briand L, Vaz E (2019) A machine learning-based approach for demarcating requirements in textual specifications. In: Proceedings of the 27th IEEE international requirements engineering conference (RE'19)

Aggarwal CC (2018) Machine learning for text. Springer

Apache OpenNLP (2017) Apache OpenNLP. http://opennlp.apache.org, last accessed: September 2019

Arora C, Sabetzadeh M, Briand L, Zimmer F (2015) Automated checking of conformance to requirements templates using natural language processing. IEEE Trans Soft Eng (TSE) 41(10):944–968

Arora C, Sabetzadeh M, Briand L, Zimmer F (2016) Extracting domain models from natural-language requirements: Approach and industrial evaluation. In: Proceedings of the 19th international conference on model driven engineering languages and systems (MODELS'16), pp 250–260

Arora C, Sabetzadeh M, Briand L, Zimmer F (2017) Automated extraction and clustering of requirements glossary terms. IEEE Trans Soft Eng (TSE) 43(10):918–945

Arora C, Sabetzadeh M, Nejati S, Briand L (2019) An active learning approach for improving the accuracy of automated domain model extraction. ACM Trans Soft Eng Method (TOSEM) 28(1):4,1–4,34

AsposeWords (2018) Java word documents manipulation APIs. https://products.aspose.com/words/java, last accessed: March 2019

Asuncion H, Asuncion A, Taylor R (2010) Software traceability with topic modeling. In: Proceedings of the 32nd international conference on software engineering (ICSE'10), pp 95–104

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res (JMLR) 13(1):281–305

Berry D, Kamsties E, Krieger M (2003) From contract drafting to software specification: Linguistic sources of ambiguity, a handbook. http://se.uwaterloo.ca/dberry/handbook/ambiguityHandbook.pdf last accessed: March 2019

Berry DM (2017) Evaluation of tools for hairy requirements and software engineering tasks. In: Proceedings of the 25th international requirements engineering conference workshops (REW'17), pp 284–291

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

Casamayor A, Godoy D, Campo M (2010) Identification of non-functional requirements in textual specifications: a semi-supervised learning approach. Information and Software Technology (IST) 52(4):436–445
Eckart de Castilho R, Gurevych I (2014) A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In: Proceedings of the workshop on open infrastructures and analysis frameworks for HLT (OIAF4HLT'14), pp 1–11
Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requirements Engineering Journal (RE J) 12(2):103–120
Cleland-Huang J, Czauderna A, Gibiec M, Emenecker J (2010) A machine learning approach for tracing regulatory codes to product specific requirements. In: Proceedings of the 32nd international conference on software engineering (ICSE'10), pp 155–164
Cohen J (1960) A coefficient of agreement for nominal scales. Educational and Psychological Measurement (EPM) 20(1):37–46
Cook WA (1989) Case grammar theory. Georgetown University Press
Covington MA (2001) A fundamental algorithm for dependency parsing. In: Proceedings of the 39th annual ACM southeast conference, Citeseer, pp 95–102
Cunningham H, Tablan V, Roberts A, Bontcheva K (2013) Getting more out of biomedical documents with gate's full lifecycle open source text analytics. PLoS Comput Bio 9(2):e1002,854
Dalpiaz F, DellÕAnna D, Aydemir FB, Çevikol S (2019) Requirements classification with interpretable machine learning and dependency parsing. In: Proceedings of the 27th IEEE international requirements engineering conference (RE'19)
Dietterich TG (1998) Approximate statistical test for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1923
Falkner A, Palomares C, Franch X, Schenner G, Aznar P, Schoerghuber A (2019) Identifying requirements in requests for proposal: A research preview. In: Proceedings of the 25th international working conference on requirements engineering: foundation for software quality (REFSQ'19), pp 176–182
Fetzer A, Johansson M (2010) Cognitive verbs in context: a contrastive analysis of english and french argumentative discourse. International Journal of Corpus Linguistics 15(2):240–266
Geller J, Kitano H, Suttner CB (1997) Parallel Processing for Artificial Intelligence 3. Elsevier
Goodfellow I, Bengio Y, Courville A (2016) Deep Learning, 1st edn. MIT Press
Guo J, Cheng J, Cleland-Huang J (2017) Semantically enhanced software traceability using deep learning techniques. In: Proceedings of the 39th international conference on software engineering (ICSE'17), pp 255–272
Guzman E, Ibrahim M, Glinz M (2017) A little bird told me: Mining tweets for requirements and software evolution. In: Proceedings of the 25th international requirements engineering conference (RE'17), pp 11–20
Habernal I, Gurevych I (2017) Argumentation mining in user-generated web discourse. Computational Linguistics 43(1):125–179
Habernal I, Eckle-Kohler J, Gurevych I (2014) Argumentation mining on the web from information seeking perspective. In: Proceedings of the workshop on frontiers and connections between argumentation theory and natural language processing (ArgNLP'14)
Hirschberg J, Manning CD (2015) Advances in natural language processing. Science 349(6245):261–266
IBM DOORS (2020) IBM - Rational DOORS. https://www.ibm.com/us-en/marketplace/requirements-management, last accessed: March 2020
Indurkhya N, Damerau FJ (2010) Handbook of Natural Language Processing, 2nd edn. CRC Press
International Organization for Standardization (2011) ISO/IEC/IEEE 29148:2011 - Systems and software engineering - Requirements engineering
Jurafsky D, Martin J (2009) Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd edn. Prentice Hall
Kurtanović Z, Maalej W (2017) Automatically classifying functional and non-functional requirements using supervised machine learning. In: Proceedings of the 25th international requirements engineering conference (RE'17), pp 490–495
Kurtanovic Z, Maalej W (2017) Mining user rationale from software reviews. In: Proceedings of the 25th international requirements engineering conference (RE'17), pp 61–70
van Lamsweerde A (2009) Requirements Engineering: From System Goals to UML Models to Software Specifications, 1st edn. Wiley
Li C, Huang L, Ge J, Luo B, Ng V (2018) Automatically classifying user requests in crowdsourcing requirements engineering. J Syst Softw (JSS) 138(1):108–123
Likert R (1932) A technique for the measurement of attitudes. Archives of Psychology 22:140
Louppe G (2014) Understanding random forests. Cornell University Library
Louridas P, Ebert C (2016) Machine learning. IEEE Softw 33(5):110–115

Maalej W, Kurtanović Z, Nabil H, Stanik C (2016) On the automatic classification of app reviews. Requirements Engineering Journal (RE J) 21(3):311–331
Manning C, Raghavan P, Schütze H (2008) Introduction to Information Retrieval. Cambridge
Manning CD, Surdeanu M, Bauer J, Finkel JR, Bethard S, Mcclosky D (2014) The Stanford coreNLP natural language processing toolkit. In: 52Nd annual meeting of the association for computational linguistics (ACL): system demonstrations, ACL, Baltimore, USA, pp 55-60
McHugh ML (2012) Interrater reliability: the kappa statistic. Biochemia Medica (BM) 22(3):276–282
Mich L, Franch M, Novi Inverardi PL (2004) Market research for requirements analysis using linguistic tools. Requirements Engineering Journal (RE J) 9(1):40–56
Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th international neural information processing systems conference (NIPS'13), pp 3111–3119
Miller GA (1995) Wordnet: a lexical database for English. Commun ACM 38(11):39–41
Nivre J, Hall J, Nilsson J (2006) MaltParser: A data-driven parser-generator for dependency parsing. In: Proceedings of the 5th international conference on language resources and evaluation (LREC'06), pp 2216–2219
Ott D (2013) Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In: Proceedings of the 19th international working conference on requirements engineering: foundation for software quality (REFSQ'13), pp 50–64
Perini A, Susi A, Avesani P (2013) A machine learning approach to software requirements prioritization. IEEE Trans Softw Eng (TSE) 39(4):445–461
Petrov S, Barrett L, Thibaux R, Klein D (2006) Learning accurate, compact, and interpretable tree annotation. In: Proceedings of the 21st international conference on computational linguistics (COLING'06), pp 433–440
Pohl K (2010) Requirements Engineering - Fundamentals, Principles, and Techniques, 1st edn. Springer
Pohl K, Rupp C (2011) Requirements Engineering Fundamentals, 1st edn. Rocky Nook
Princeton University (2010) About WordNet. https://wordnet.princeton.edu/documentation, last accessed: March 2019
Ramshaw L, Marcus M (1999) Text chunking using transformation-based learning. In: Natural language processing using very large corpora, Springer
Reutemann P, van Rijn J, Frank E (2018) Weka MultiSearch Parameter Optimization. http://weka.sourceforge.net/packageMetaData/multisearch/index.html last accessed:, March 2019
Riaz M, King J, Slankas J, Williams L (2014) Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In: Proceedings of the 22nd international requirements engineering conference (RE'14), pp 183–192
Rodeghero P, Jiang S, Armaly A, McMillan C (2017) Detecting user story information in developer-client conversations to generate extractive summaries. In: Proceedings of the 39th international conference on software engineering (ICSE'17), pp 49–59
Rolland C, Proix C (1992) A natural language approach for requirements engineering. In: International conference on advanced information systems engineering, Springer, pp 257–277
Stamatatos E (2009) A survey of modern authorship attribution methods. Journal of the American Society for information Science and Technology 60(3):538–556
Sultanov H, Hayes JH (2013) Application of reinforcement learning to requirements engineering: requirements tracing. In: Proceedings of the 21st international requirements engineering conference (RE'13), pp 52–61
Suthaharan S (2016) Modeling and algorithms. Springer, USA, pp 123–143
Walenz B, Didion J (2011) JWNL: Java WordNet Library. http://jwordnet.sourceforge.net last accessed: March 2019
Wang F, Yang Z, Huang Z, Liu C, Zhou Y, Bodeveix J, Filali M (2019) An approach to generate the traceability between restricted natural language requirements and aadl models. IEEE Trans Reliab, pp 1–20
Williams G, Mahmoud A (2017) Mining twitter feeds for software user requirements. In: Proceedings of the 25th international requirements engineering conference (RE'17), pp 1–10
Winkler J, Vogelsang A (2016) Automatic classification of requirements based on convolutional neural networks. In: Proceedings of the 24th international requirements engineering conference workshops (REW'16), pp 39–45
Winkler J, Vogelsang A (2018) Using tools to assist identification of non-requirements in requirements specifications–a controlled experiment. In: Proceedings of the 24th international working conference on requirements engineering: foundation for software quality (REFSQ'18), pp 57–71

Winkler JP, Grönberg J, Vogelsang A (2019) Optimizing for recall in automatic requirements classification: An empirical study. In: Proceedings of the 27th IEEE international requirements engineering conference (RE'19)

Witten IH, Frank E, Hall MA, Pal CJ (2016) Data Mining: Practical Machine Learning Tools and Techniques, 4th edn. Morgan Kaufmann

Yang H, Willis A, De Roeck A, Nuseibeh B (2010) Automatic detection of nocuous coordination ambiguities in natural language requirements. In: Proceedings of the 25th international conference on automated software engineering (ASE'10), pp 53–62

Yang H, De Roeck A, Gervasi V, Willis A, Nuseibeh B (2012) Speculative requirements: Automatic detection of uncertainty in natural language requirements. In: Proceedings of the 20th international requirements engineering conference (RE'12), pp 11–20

Zhao L, Alhoshan W, Ferrari A, Letsholo KJ, Ajagbe MA, Chioasca E (2020) Batista-navarro RT Natural language processing (NLP) for requirements engineering: A systematic mapping study. arXiv:2004.01099

Sallam Abualhaija is a research associate at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. Before that, Abualhaija worked as a postdoctoral researcher at L3S research center in Hannover (Germany). Abualhaija holds a Ph.D. in Computational Linguistics from Hamburg University of Technology with a scholarship from DAAD. Her main research interests lie within the fields of applied natural language processing, text mining and machine learning with a current focus on requirements engineering and regulatory compliance. Abualhaija enriched her experience by working on a wide spectrum of research projects like social media mining in crises management in collaboration with different German emergency responders, compliance verification of legal artifacts according to European laws and quality assurance of textual requirements specifications. Abualhaija has been often on the program committee of computational linguistics conferences like EMNLP and NAACL, reviewed scientific papers for multiple journals like JSS, and has recently co-organized the NLP4RE workshop held in parallel with the REFSQ conference.

Chetan Arora is a Senior Lecturer in Software Engineering at the Deakin University. Prior to this position, he was working in Innovation Programs at SES Satellites in Luxembourg on project and technical management across several EU-H2020 and European Space Agency (ESA) projects on IoT and SatCom for disaster management, and applications of AI in SatCom. He received his Ph.D. degree from the University of Luxembourg (Luxembourg) in 2016. Over the last decade, Arora has been working in applied research in collaboration with several industry and government sectors, including satellite communication, automotive and crisis management. His research interests include requirements engineering, IoT, empirical software engineering, and applied machine learning and natural language processing.

Mehrdad Sabetzadeh is an Associate Professor at the School of Electrical Engineering and Computer Science of the University of Ottawa and a part-time Faculty Member at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. Previously, Sabetzadeh worked as a permanent member of the research staff at Simula Research Laboratory (Norway), and as an NSERC postdoctoral fellow at University College London (UK). Sabetzadeh received his Ph.D. in Computer Science from the University of Toronto. His main research interests are in software engineering with an emphasis on requirements engineering, model-based development, and regulatory compliance. Sabetzadeh is passionate about fostering stronger ties between academia and industry; in the past decade, he has conducted most of his research in close collaboration with industry partners. His experience spans several sectors, including government, finance, legal services, telecommunications, maritime, energy, aerospace, railways, and automotive. Sabetzadeh has co-authored more than 70 scientific papers and secured more than $6M of research funding as lead investigator. He has been on the organizing or program committees of several international conferences such as RE, ICSE, ESEC/FSE, and MODELS.

Lionel C. Briand is professor of software engineering and has shared appointments between (1) School of Electrical Engineering and Computer Science, University of Ottawa, Canada and (2) The SnT centre for Security, Reliability, and Trust, University of Luxembourg. He is the head of the SVV department at the SnT Centre and a Canada Research Chair in Intelligent Software Dependability and Compliance (Tier 1). He holds an ERC Advanced Grant, the most prestigious European individual research award, and has conducted applied research in collaboration with industry for more than 25 years, including projects in the automotive, aerospace, manufacturing, financial, and energy domains. In 2010, he was elevated to the grade of IEEE fellow for his work on testing of object-oriented systems. He was also granted the IEEE Computer Society Harlan Mills award (2012) and the IEEE Reliability Society Engineer-of-the-year award (2013) for his work on model-based verification and testing. His research interests include: Model-driven development, testing and verification, search-based software engineering, requirements engineering, and empirical software engineering.

Michael Traynor received his master's in computer science at Dalhousie University before working as a data scientist at QRA. His research interests have focused on deep learning and natural language processing, especially on character-based language models. However, he enjoys keeping up on any and all things related to artificial intelligence.

Affiliations

Sallam Abualhaija[1] · Chetan Arora[1,2] · Mehrdad Sabetzadeh[1,3] · Lionel C. Briand[1,3] · Michael Traynor[4]

Chetan Arora
chetan.arora@deakin.edu.au

Mehrdad Sabetzadeh
m.sabetzadeh@uottawa.ca

Lionel C. Briand
lbriand@uottawa.ca

Michael Traynor
miket@qracorp.com

1 SnT Centre for Security, Reliability, and Trust, University of Luxembourg, Alzette, Luxembourg

2 School of Information Technology, Deakin University, Geelong, Australia

3 School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada

4 QRA Corp, Halifax, Canada