A Machine Learning-Based Approach for Demarcating Requirements in Textual Specifications

Sallam Abualhaija*, Chetan Arora*, Mehrdad Sabetzadeh*, Lionel C. Briand*†, Eduardo Vaz‡
*SnT Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
†School of Engineering and Computer Science, University of Ottawa, Canada
‡QRA Corp, Halifax, Canada
Email: {abualhaija, arora, sabetzadeh, briand}@svv.lu, eduardo@qracorp.com

***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 30 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.

***Index Terms*—Textual Requirements, Requirements Identification and Classification, Machine Learning, Natural Language Processing.**

I. Introduction

Requirements specifications (RS) 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 [1]. RS 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, RS are predominantly written in natural language [2], [3], [4].

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 [1], [4]. 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 [5] – is important for multiple reasons: First, requirements are typically the basis for development contracts [6]. 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 [2], [7]. 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 [4].

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. This is because, 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.

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

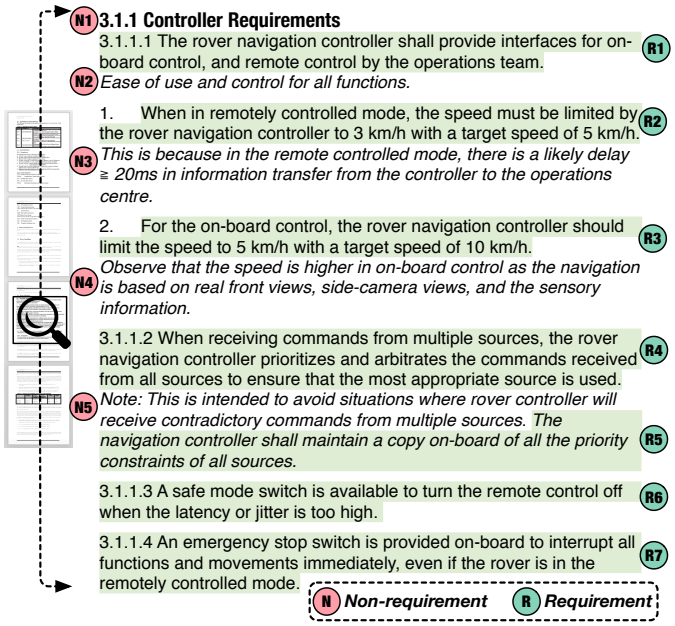

Fig. 1: Excerpt from a Textual Requirements Specification.

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.

In this paper, we develop a practical, accurate and fully automated approach for *requirements demarcation* in textual RS. 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 RS, particularly customer-provided ones, in search of requirements. As such, they have shown interest in a tool feature that would help them identify the requirements more efficiently.

Much work has been done in the RE community on automating the identification and classification of requirements-related information in textual documents [5], [8]. However, and as we argue more precisely in Section V, 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 RS written in a variety of styles.

Contributions. We employ machine learning (ML) to devise an automated approach for requirement demarcation. Our ML classification model is based on generic features that can be applied to a wide range of RS 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) RS; 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 RS in practice [3]. While our approach is not limited to free-form RS, it is not necessarily optimized for structured RS, 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 RS are sufficiently structured; for such RS, 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 30 industrial RS from 12 different application domains and originating from 18 different organizations. These RS, which were labeled either directly by our industry partner or by an independent, trained annotator (non-author), collectively contain 7351 requirements and 10955 non-requirements. We exploit this dataset in an empirically sound manner 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.

Structure. Section II provides background. Section III presents our approach. Section IV describes our empirical evaluation. Section V compares with related work. Section VI discusses threats to validity. Section VII concludes the paper.

II. Background

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

A. Machine Learning

Our approach is based on *supervised* ML, meaning that it requires labeled data for training. Our labeled data is made up of RS whose different segments have been marked by human experts as being requirements or non-requirements. Supervised techniques are categorized into *classification* and *regression* [9]; 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 IV), 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. However, in many situations, like ours, the costs associated with different misclassification types are not symmetric. In particular, 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 completeness, i.e., the ability to identify all the requirements.

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* [9]. In our approach, we employ *cost-sensitive learning*, which is generally considered to be the more effective of the two [10]. 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 underrepresented in our dataset (see Section IV). We use cost-sensitive learning exclusively for mitigating, as much as possible, false negatives.

B. Natural Language Processing

Natural language processing (NLP) is concerned with the computerized understanding, analysis, and production of human-language content [11], [12]. In this paper, 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 [13] for information about words' senses (meanings) and the relations between these senses. In this paper, 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 [14]. 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 from products and software reviews [15], [16]; including action and stative verbs is motivated by Fillmore's basic English verb types [17].

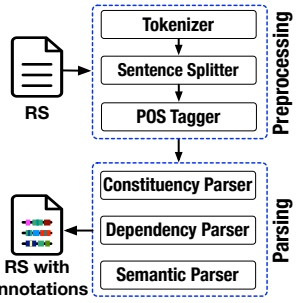

Fig. 2: Our NLP Pipeline.

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 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.

III. Approach

Fig. 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 paper, given the binary nature of classification, we refer to each sentence as a *requirement candidate*, or cand for short. The output of the approach is a demarcated RS; this is the input RS where each cand has been marked as either a requirement or a non-requirement.

The approach works in three phases, labeled 1–3 in Fig. 3. In the first phase, we run the NLP pipeline of Fig. 2 on the

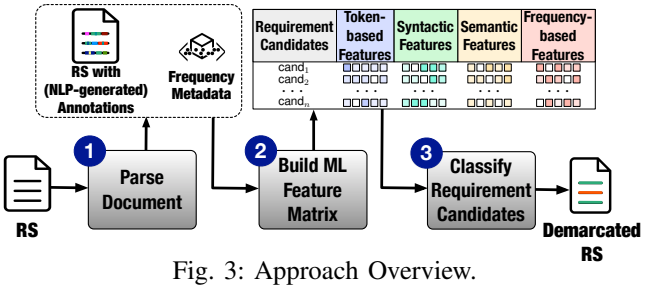

Fig. 3: Approach Overview.

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 frequency information obtained from the first phase. In the third and final phase, we perform the classification using ML. The rest of this section elaborates the three phases of our approach.

A. 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 cands, with the other modules in the pipeline providing part-of-speech, syntactic and semantic annotations for the cands. 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 lowercase 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 RS. We are interested in the id patterns and their frequencies due to the following general intuition: A cand 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 Frq1 in Table I).

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

B. Building an ML Feature Matrix

Our feature design has been driven by the need to keep the features generic; this means that 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 irrespectively of the exact characteristics of individual RS. Table I lists our features. These features are computed for every cand. For each feature, the table provides an id, short name, type, description, intuition, and one or more examples. We organize the features into four categories:

- The *token-based features* (Tok1–Tok6) are based on the token-level information in a cand.
- The *syntactic features* (Syn1–Syn8) are derived from the syntax-related information in a cand, 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 cand. These categories were defined in Section II-B.
- The *frequency-based features* (Frq1–Frq3) are derived for each cand based on the document-wide frequency metadata discussed in Section III-A as well as the syntax-related information within that particular cand.

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

C. 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 RS (see Section IV). 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 cands in each RS, and (3) merging the (labeled) data from the individual RS 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 next.

IV. Evaluation

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

A. 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.

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

RQ3. How useful is our approach in practice? Building on the result of RQ1, RQ3 assesses the accuracy of our approach over previously unseen RS.

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 RS 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

TABLE I: Features for Learning.

	ID	Short Name (N), Type (T), Description (D), Intuition (I) and Example (E)
Token-based	Tok1	**(N)** numTokens **(T)** Numeric **(D)** Number of tokens in a cand*. **(I)** A cand that is too long or too short could indicate a non-requirement. **(E)** The tokenizer returns 7 tokens for N1 in Fig. 1: {"3", ".", "1", ".", "1", "Controller", "Requirements"}. Therefore, Tok1(N1) = 7.
	Tok2	**(N)** numAlphabetics **(T)** Numeric **(D)** Number of alphabetic words in a cand. **(I)** Few alphabetic words in a cand could indicate a non-requirement. **(E)** Tok2(N1) = 2 because there are only two alphabetic tokens in N1, namely "Controller" and "Requirements".
	Tok3	**(N)** numOneCharTokens **(T)** Numeric **(D)** Number of one-character tokens in a cand. **(I)** Too many one-character tokens in a cand could indicate a non-requirement, e.g., section headings. **(E)** Tok3(N1) = 5.
	Tok4	**(N)** startsWithId **(T)** Boolean **(D)** TRUE if a cand 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. **(E)** Tok4(R1) = TRUE.
	Tok5	**(N)** startsWithTriggerWord **(T)** Boolean **(D)** TRUE if a cand begins with a trigger word ("Note", "Rationale", "Comment"), otherwise FALSE. **(I)** A trigger word at the beginning of a cand is a strong indicator for a non-requirement. **(E)** N5 begins with "Note", thus Tok5(N5) = TRUE.
	Tok6	**(N)** hasUnits **(T)** Boolean **(D)** TRUE if a cand 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 cand being a requirement. **(E)** Tok6(R2) = TRUE because of the km/h unit appearing in R2.
Syntactic	Syn1	**(N)** hasVerb **(T)** Boolean **(D)** TRUE if a cand has a verb as per the POS tags, otherwise FALSE. **(I)** A cand without a verb is unlikely to be a requirement. **(E)** Syn1(N1) = FALSE, whereas Syn1(R1) = TRUE.
	Syn2	**(N)** hasModalVerb **(T)** Boolean **(D)** TRUE if a cand has a modal verb, otherwise FALSE. **(I)** The presence of a modal verb is a good indicator for a cand being a requirement. **(E)** Syn2(R2) = TRUE because of the modal verb "must".
	Syn3	**(N)** hasNPModalVP **(T)** Boolean **(D)** TRUE if a cand contains a sequence composed of a Noun Phrase (NP) followed by a Verb Phrase (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 preceeding a modal VP. This NP typically acts as a subject for the VP. **(E)** The NP-followed-by-Modal-VP pattern has a match in R3: "The rover navigation controller should limit […]". Thus, Syn3(R3) = TRUE.
	Syn4	**(N)** startsWithDetVerb **(T)** Boolean **(D)** TRUE if a cand, excluding head alphanumeric patterns / triggers 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. **(E)** Syn4(N3) = TRUE. Also, Syn4(N5) = TRUE because, when "Note:" is excluded from the head of N5, the remainder begins with a determiner followed by a verb.
	Syn5	**(N)** hasConditionals **(T)** Boolean **(D)** TRUE if a cand has a conditional clause, otherwise FALSE. **(I)** Conditional clauses are more likely to appear in requirements than non-requirements. **(E)** R4 has a conditional clause: "When receiving commands from multiple sources". Thus, Syn5(R4) =TRUE.
	Syn6	**(N)** hasPassiveVoice **(T)** Boolean **(D)** TRUE if a cand has passive voice through some dependency relation, otherwise FALSE. **(I)** Requirements not containing modal verbs may be specified in passive voice. **(E)** Syn6(R7) = TRUE.
	Syn7	**(N)** hasVBToBeAdj **(T)** Boolean **(D)** TRUE if, in cand, 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. **(E)** Syn7(R6) = TRUE because it includes "is available".
	Syn8	**(N)** isPresentTense **(T)** Boolean **(D)** TRUE if a cand has some root verb which is in present tense, otherwise FALSE. **(I)** Sometimes, requirements are written in present tense rather than with modal verbs. **(E)** Syn8(R4) = TRUE; the root verb is "prioritizes".
Semantic	Sem1	**(N)** hasCognitionVerb **(T)** Boolean **(D)** TRUE if a cand has some verb conveying reasoning or intention, otherwise FALSE. **(I)** Reasoning and intention are a common characteristic of non-requirements. **(E)** Sem1(N5) = TRUE because of the verb "intended".
	Sem2	**(N)** hasActionVerb **(T)** Boolean **(D)** TRUE if a cand has some verb conveying motion or change of status, otherwise FALSE. **(I)** Action verbs are common in requirements for describing behaviors and state changes. **(E)** Sem2(R3) = TRUE because of the verb "limit".
	Sem3	**(N)** hasStativeVerb **(T)** Boolean **(D)** TRUE if a cand has some stative verb, otherwise FALSE. **(I)** Stative verbs are common in requirements for describing system properties. **(E)** Sem3(R5) = TRUE because of the verb "maintain".
Frequency-based	Frq1	**(N)** idPatternFrequency **(T)** Enumeration **(D)** Maximum frequency level (high, medium, low) associated with the identifier pattern with which a given cand starts. If a cand does not start with an alphanumeric pattern, the returned value is NA (not applicable). **(I)** A frequent id pattern in a cand is likely to signify a requirement. This is because alphanumerics are prevalently used for marking requirements. **(E)** For the excerpt of Fig. 1: Frq1(R1) = medium, Frq1(R2) = low, Frq1(N1) = low, and Frq1(R5) = NA.
	Frq2	**(N)** hasMFModalVerb **(T)** Boolean **(D)** TRUE if a cand 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. **(E)** "Shall" is the most frequent modal verb in the excerpt of Fig. 1. Thus, Frq2(R1) = TRUE and Frq2(R2) = FALSE.
	Frq3	**(N)** hasHFNP **(T)** Boolean **(D)** TRUE if a cand contains some 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. **(E)** The most frequent NP in Fig. 1 is "rover navigation controller". Thus, Frq3(R1) = TRUE.

* *The term "cand" is an abbreviation for "requirement candidate".*

in return. This enables us to determine whether we can leave out some expensive-to-compute features without significantly degrading the quality of demarcation.

B. Implementation

We have implemented our demarcation approach in Java. The implementation is ≈7500 lines of code, excluding comments and third-party libraries. We use Aspose [18] for extracting the textual content of RS (provided as MS Word documents) and DKPro [19] for operationalizing the NLP pipeline (Fig. 2). Our operationalization of the pipeline employs Berkeley [20] for constituency parsing, Malt [21] for dependency parsing, and JWNL (the Java WordNet Library) [22] for semantic parsing. ML classification is done using Weka [10].

C. Data Collection and Preparation

Our data collection focused on procuring a representative set of free-form RS and manually demarcating these RS. We gathered 30 industrial RS from 12 application domains, including, among others, information systems, automotive, healthcare,

aviation, aerospace, telecommunications, and networks. These RS originate from a total of 18 different organizations.

Among the 30 RS, 12 had their requirements already marked by our industry partner in collaboration with the respective system clients. The requirements in the remaining 18 RS 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 RS, 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 [23] and the relevant chapters of two requirements engineering textbooks [1], [4].

In the next step and using the NLP Sentence Splitter module discussed in Section II, we transformed each RS into a set of sentences. We recall from Section III that these sentences, referred to as requirement candidates (cands, for short), are the units for classification. Applying sentence splitting to the 30 RS resulted in a total of 18306 cands. We mapped these cands onto the manually identified requirement regions as per the annotations provided by our industry partner and third-party annotator. Specifically, any cand whose span intersected with a (manually specified) requirement region was deemed as a requirement. All other cands were deemed as non-requirements. This process led to 7351 cands marked as requirements and 10955 cands 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 RS 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 RS and contained ≈2200 cands. The interrater agreement between the annotator and the two researchers was computed using Cohen's Kappa (κ) [24]. A cand 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") [25]. The disagreements were almost exclusively over cands 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 cands 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 cands 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 cands in E to be as close as possible to 10% of the total number of cands. To further increase the validity and generalizability of our results, we enforced the following additional criteria: (1) The RS in E must all have distinct origins (sources) and distinct application domains, (2) The RS in E must have different origins than those in T. Our split strategy resulted in an E with four RS, RS1 to RS4, as described in Table II. We refer back to these RS when answering RQ3. The four RS contain 2145 cands, of which 663 are requirements and 1482 are non-requirements. These RS constitute 11.7% of the cands in our dataset (9% of all requirements and 13.5% of all non-requirements).

TABLE II: The RS in our Validation Set (E).

Id	Description of the RS	# of Cands Marked as Requirement	# of Cands Marked as Non-Requirement
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

D. Metrics for the Evaluation of ML Classification Models

We use standard metrics, *Accuracy (A)*, *Precision (P)* and *Recall (R)* [10], for evaluating ML classification models. *Accuracy* is computed as the ratio of cands correctly demarcated as requirement and non-requirement to the total number of cands. *Precision* is the ratio of cands correctly classified as requirement to all cands 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 [26].

E. 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 [27]. In our evaluation, we considered, from each of these three categories, widely used and recommended algorithms [28]. 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 [29], [30]. 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

TABLE III: ML Algorithm Selection Results (RQ1).

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

A(%) = Accuracy (in percentage) | P(%) = Precision (in percentage) | R(%) = Recall (in percentage)

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 II-A. 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 I using information gain (IG) [10] 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 RS in T and applying the resulting classification model for demarcating the RS 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 cand containing a modal verb; **(B2)** marking as requirement any cand containing the most frequent modal verb of the underlying RS; **(B3)** marking as requirement any cand 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 IV-A (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 ($T \cup E$).

F. Discussion

Below, we answer the RQs posed in Section IV-A.

RQ1. Table III 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 III, we conclude that **Random Forest presents a slight or moderate advantage over the alternatives** across all three metrics. We answer the remaining RQs using Random Forest as our classification algorithm.

RQ2. Fig. 4 lists the features of Table I in descending order of information gain (IG). Based on the results in this figure, **the most influential features are Frq2, Syn2, Syn3, Syn1, Tok2, and Tok1.** The top-three features – Frq2, Syn2 and Syn3 – 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 – Syn1, Tok2 and Tok1 – are targeted at excluding non-requirements. With the exception of Tok4 and Syn4, 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 Tok4 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 Frq1. The second and last feature with an IG score of zero is Syn4, indicating that the linguistic pattern represented by this feature does not contribute to distinguishing requirements from non-requirements. In summary, the results of Fig. 4 provide evidence that all but two of our features (Tok4 and Syn4) indeed play a role in requirements demarcation, thus confirming the majority of the intuitions presented in Table I.

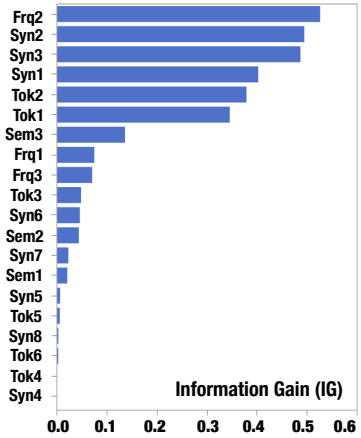

Fig. 4: Feature Importance (RQ2).

RQ3. Table IV 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

TABLE IV: Model Validation Results (RQ3).

	RS1			RS2			RS3			RS4		
	A(%)	P(%)	R(%)	A(%)	P(%)	R(%)	A(%)	P(%)	R(%)	A(%)	P(%)	R(%)
RF (¬CSL)	95.6	**100**	85.7	**92.4**	85.4	88.5	**92.1**	82.5	87.3	87.5	84.6	82.8
RF (CSL)	**99.7**	**100**	**99.1**	88.1	71.2	**97.8**	89.6	72.8	**94.2**	**88.3**	80.6	**91.6**
B1	63.8	44.0	62.5	85.8	81.3	65.5	84.3	70.6	65.3	67.3	57.3	62.3
B2	81.2	92.3	42.9	84.4	**97.0**	46.8	87.1	**91.3**	54.3	82.4	**97.8**	56.0
B3	82.0	66.9	83.0	52.8	29.0	45.3	81.8	62.1	72.8	79.2	71.8	76.6
B4	58.6	41.8	86.6	61.6	41.7	86.3	79.8	56.5	88.4	64.7	52.9	84.1
B5	80.9	65.0	83.0	61.0	40.2	74.8	83.9	63.5	85.5	81.1	72.9	82.0

applying the learned classification model to the validation set (E), i.e., RS1–RS4 in Table II. 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 IV-E (*Comparison with Baselines*).

We observe from Table IV 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 RS (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 RS (RS2) than our classification model with cost-sensitive learning (difference of 10.1%); nevertheless, **B1**'s recall is much lower (difference of 32.3%).

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, Syn2 and Syn3 relate to **B1**, Frq2 to **B2**, and Tok4 and Frq1 to **B3**. As we discussed in RQ2, all our features except Tok4 and Syn4 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 IV-E, 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 paper, **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 the requirements not missed. **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.**

TABLE V: Root Causes for False Negatives.

	Cause	Explanation and Example	Count
1	**Loss of Context**	The units of analysis in our approach are sentences. A requirement that is specified over multiple sentences, including enumerated-list items, would thus constitute multiple classification units. Sometimes, the context is lost after the first unit within such requirements, resulting in the misclassification of the remaining units. **Example:** In the list below, we have three sentences (cands). *The control interface shall be tolerant and function:* $cand_1$ *1) if the round-trip latency < 250ms;* $cand_2$ *2) if the jitter < 20% of the average latency.* $cand_3$ We correctly demarcate $cand_1$ as a requirement, but miss $cand_2$ and $cand_3$ (two false negatives)	20
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 angle (REF27 in SD119) for the controller should, upon ACK, be set to 34°.*"	8
3	**Conservative Ground Truth**	As noted in Section IV-C, our third-party annotator was instructed to favor the requirement class when in doubt. In our error analysis, we observed borderline situations where the automated classification could well be correct, but did not match the deliberately conservative ground truth. **Example:** The following is a requirement in our ground truth but is classified as a non-requirement by our approach. *"The decision to retain any single-point failures of any severity level in the design is subject to formal approvals on a case-by-case basis, with a detailed analysis for each failure."*	6

When employed with cost-sensitive learning, our approach misclassifies as non-requirement a total of 34 requirements across RS1–RS4. We analyzed all the misclassifications in order to determine the root causes. We identified three root causes, as shown in Table V. 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 20 cases in row 1, our approach demarcated only the first sentence or the enumerated-list header of a requirement. In practice, when an analyst reviews the automatically generated demarcations, these incomplete cases are relatively easy to spot and fix. 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 III. In the case of the provider, these two phases will be executed on a large corpus of *already annotated* RS 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.

TABLE VI: Demarcation Quality vs. Execution Time (RQ5).

	Feature Group (Execution Time in Milliseconds per Cand (ms/c))														
	Tok (1.2 ms/c)			TokSynFrq (101 ms/c)			TokSem (2 ms/c)			All (102 ms/c)			All-{Syn5} (49 ms/c)		
RF	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)
(CSL)	79.9	67.6	95.7	93.5	88.4	96.4	80.9	68.8	96.0	**93.8**	**88.9**	**96.5**	93.6	88.6	96.4

Best Tradeoff

Over our training set (T) which is composed of 16161 cands, the first two phases of the approach took 27.3 minutes to run, i.e., an average of ≈101 milliseconds per cand. Training the classification model took negligible time (16 seconds). **This execution 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 RS collectively contain 2145 cands, giving an average processing time of ≈105 milliseconds per cand. 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 cands 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.

RQ5. To compute the token-based features of Table I, 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 VI, 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 cand 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 (Frq2 and Frq3), one can replace constituency parsing with text chunking (shallow parsing) [31]. 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 [32], [33]. Based on the RQ2 results, Syn5 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 VI, we reduce the execution time from 102 to 49 milliseconds per cand 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 cands per page, our tradeoff solution reduces the execution time from 3 seconds to 1.5 seconds per page.

V. Related Work

ML has been utilized as a way to provide computerized assistance for several requirements engineering tasks, e.g., trace link generation [34], [35], [36], [37], requirements identification and classification [38], [39], [40], prioritization [41], ambiguity detection [42], [43], relevance analysis [44], and review classification [45], [15]. The application of ML over textual requirements is almost always preceded by some form of NLP. 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 [39], [5] propose an approach based on deep learning [9] for addressing the same problem that we address: requirements demarcation. They train their classifier on word embeddings [46] 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 [47]. 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 RS, 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. [8] 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 [48] 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. [38] 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. [49], Riaz et al. [50], and Li et al. [51] propose similar techniques based on keywords to predict categories for different requirements. Guzman et al. [52] and Williams and Mahmoud [53] mine requirements from twitter feeds through a combination of ML and NLP preprocessing. Rodeghero et al. [54] 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 [40] 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. 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 IV indicate (see RQ2), all feature types are influential for an accurate differentiation of requirements and non-requirements.

VI. Threats to Validity

The validity dimensions most pertinent to our evaluation are internal, construct and external validity.

Internal Validity. Bias was the main internal validity threat that we needed to counter. To mitigate bias risks, the manual classification of our dataset, as we discussed in Section IV-C, 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 RS in the validation set other than the application domains and the numbers of requirements and non-requirements in these RS; this minimal information about the validation set was necessary for planning our experimental procedures (see Section IV-E).

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 IV-C, 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 II-A. 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 evaluation was based on a relatively large dataset with the RS 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 RS 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 empirical evaluations.

VII. 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 30 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.

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.

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

REFERENCES

[1] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 1st ed. Wiley, 2009.

[2] D. Berry, E. Kamsties, and M. Krieger, "From contract drafting to software specification: Linguistic sources of ambiguity, a handbook," 2003, last accessed: March 2019. [Online]. Available: http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

[3] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering Journal (RE J)*, vol. 9, no. 1, pp. 40–56, 2004.

[4] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*, 1st ed. Springer, 2010.

[5] J. Winkler and A. Vogelsang, "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)*, 2018, pp. 57–71.

[6] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer., "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)*, 2016, pp. 250–260.

[7] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals*, 1st ed. Rocky Nook, 2011.

[8] A. Falkner, C. Palomares, X. Franch, G. Schenner, P. Aznar, and A. Schoerghuber, "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)*, 2019, pp. 176–182.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. MIT Press, 2016.

[10] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Morgan Kaufmann, 2016.

[11] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Prentice Hall, 2009.

[12] N. Indurkhya and F. J. Damerau, *Handbook of Natural Language Processing*, 2nd ed. CRC Press, 2010.

[13] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[14] Princeton University, "About WordNet," 2010, last accessed: March 2019. [Online]. Available: https://wordnet.princeton.edu/documentation

[15] Z. Kurtanovic and W. Maalej, "Mining user rationale from software reviews," in *Proceedings of the 25th International Requirements Engineering Conference (RE'17)*, 2017, pp. 61–70.

[16] I. Habernal, J. Eckle-Kohler, and I. Gurevych, "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)*, 2014.

[17] W. A. Cook, *Case grammar theory*. Georgetown University Press, 1989.

[18] Aspose.Words, "Java word documents manipulation APIs," 2018, last accessed: March 2019. [Online]. Available: https://products.aspose.com/words/java

[19] R. Eckart de Castilho and I. Gurevych, "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)*, 2014, pp. 1–11.

[20] S. Petrov, L. Barrett, R. Thibaux, and D. Klein, "Learning accurate, compact, and interpretable tree annotation," in *Proceedings of the 21st International Conference on Computational Linguistics (COLING'06)*, 2006, pp. 433–440.

[21] J. Nivre, J. Hall, and J. Nilsson, "MaltParser: A data-driven parser-generator for dependency parsing," in *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'06)*, 2006, pp. 2216–2219.

[22] B. Walenz and J. Didion, "JWNL: Java WordNet Library," 2011, last accessed: March 2019. [Online]. Available: http://jwordnet.sourceforge.net

[23] International Organization for Standardization, "ISO/IEC/IEEE 29148:2011 - Systems and software engineering - Requirements engineering," 2011.

[24] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement (EPM)*, vol. 20, no. 1, pp. 37–46, 1960.

[25] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia Medica (BM)*, vol. 22, no. 3, pp. 276–282, 2012.

[26] D. M. Berry, "Evaluation of tools for hairy requirements and software engineering tasks," in *Proceedings of the 25th International Requirements Engineering Conference Workshops (REW'17)*, 2017, pp. 284–291.

[27] S. Suthaharan, *Modeling and Algorithms*. Springer US, 2016, pp. 123–143.

[28] P. Louridas and C. Ebert, "Machine learning," *IEEE Software*, vol. 33, no. 5, pp. 110–115, 2016.

[29] P. Reutemann, J. van Rijn, and E. Frank, "Weka MultiSearch Parameter Optimization," 2018, last accessed: March 2019. [Online]. Available: http://weka.sourceforge.net/packageMetaData/multisearch/index.html

[30] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research (JMLR)*, vol. 13, no. 1, pp. 281–305, 2012.

[31] L. Ramshaw and M. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora*. Springer, 1999.

[32] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated checking of conformance to requirements templates using natural language processing," *IEEE Transactions on Software Engineering (TSE)*, vol. 41, no. 10, pp. 944–968, 2015.

[33] ——, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering (TSE)*, vol. 43, no. 10, pp. 918–945, 2017.

[34] H. Asuncion, A. Asuncion, and R. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE'10)*, 2010, pp. 95–104.

[35] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE'10)*, 2010, pp. 155–164.

[36] H. Sultanov and J. H. Hayes, "Application of reinforcement learning to requirements engineering: requirements tracing," in *Proceedings of the 21st International Requirements Engineering Conference (RE'13)*, 2013, pp. 52–61.

[37] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*, 2017, pp. 255–272.

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

[39] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *Proceedings of the 24th International Requirements Engineering Conference Workshops (REW'16)*, 2016, pp. 39–45.

[40] Z. Kurtanović and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *Proceedings of the 25th International Requirements Engineering Conference (RE'17)*, 2017, pp. 490–495.

[41] A. Perini, A. Susi, and P. Avesani, "A machine learning approach to software requirements prioritization," *IEEE Transactions on Software Engineering (TSE)*, vol. 39, no. 4, pp. 445–461, 2013.

[42] H. Yang, A. Willis, A. De Roeck, and B. Nuseibeh, "Automatic detection of nocuous coordination ambiguities in natural language requirements," in *Proceedings of the 25th International Conference on Automated Software Engineering (ASE'10)*, 2010, pp. 53–62.

[43] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Speculative requirements: Automatic detection of uncertainty in natural language requirements," in *Proceedings of the 20th International Requirements Engineering Conference (RE'12)*, 2012, pp. 11–20.

[44] C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, "An active learning approach for improving the accuracy of automated domain model extraction," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 1, pp. 4:1–4:34, 2019.

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

[46] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Neural Information Processing Systems Conference (NIPS'13)*, 2013, pp. 3111–3119.

[47] IBM DOORS, "IBM - Rational DOORS," 2018, last accessed: March 2019. [Online]. Available: https://www.ibm.com/us-en/marketplace/requirements-management

[48] D. Ott, "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)*, 2013, pp. 50–64.

[49] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology (IST)*, vol. 52, no. 4, pp. 436–445, 2010.

[50] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *Proceedings of the 22nd International Requirements Engineering Conference (RE'14)*, 2014, pp. 183–192.

[51] C. Li, L. Huang, J. Ge, B. Luo, and V. Ng, "Automatically classifying user requests in crowdsourcing requirements engineering," *Journal of Systems and Software (JSS)*, vol. 138, no. 1, pp. 108–123, 2018.

[52] E. Guzman, M. Ibrahim, and M. Glinz, "A little bird told me: Mining tweets for requirements and software evolution," in *Proceedings of the 25th International Requirements Engineering Conference (RE'17)*, 2017, pp. 11–20.

[53] G. Williams and A. Mahmoud, "Mining twitter feeds for software user requirements," in *Proceedings of the 25th International Requirements Engineering Conference (RE'17)*, 2017, pp. 1–10.

[54] P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan, "Detecting user story information in developer-client conversations to generate extractive summaries," in *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*, 2017, pp. 49–59.