An Active Learning Approach for Improving the Accuracy of Automated Domain Model Extraction

CHETAN ARORA, SnT Centre for Security, Reliability and Trust, University of Luxembourg
MEHRDAD SABETZADEH, SnT Centre for Security, Reliability and Trust, University of Luxembourg
SHIVA NEJATI, SnT Centre for Security, Reliability and Trust, University of Luxembourg
LIONEL BRIAND, SnT Centre for Security, Reliability and Trust, University of Luxembourg

Domain models are a useful vehicle for making the interpretation and elaboration of natural-language requirements more precise. Advances in natural language processing (NLP) have made it possible to automatically extract from requirements most of the information that is relevant to domain model construction. However, alongside the relevant information, NLP extracts from requirements a significant amount of information that is superfluous, i.e., not relevant to the domain model. Our objective in this article is to develop automated assistance for filtering the superfluous information extracted by NLP during domain model extraction. To this end, we devise an active-learning-based approach that iteratively learns from analysts' feedback over the relevance and superfluousness of the extracted domain model elements, and uses this feedback to provide recommendations for filtering superfluous elements. We empirically evaluate our approach over three industrial case studies. Our results indicate that, once trained, our approach automatically detects an average of $\approx 45\%$ of the superfluous elements with a precision of $\approx 96\%$. Since precision is very high, the automatic recommendations made by our approach are trustworthy. Consequently, analysts can dispose of a considerable fraction – nearly half – of the superfluous elements with minimal manual work. The results are particularly promising, as they should be considered in light of the non-negligible subjectivity that is inherently tied to the notion of relevance.

CCS Concepts:•**Software and its engineering** → **Requirements analysis;** *Software design engineering; Empirical software validation;*

Additional Key Words and Phrases: Requirements Engineering, Active Learning, Natural-language Requirements, Domain Modeling, Case Study Research.

ACM Reference Format:
Chetan Arora, Mehrdad Sabetzadeh, Shiva Nejati, and Lionel Briand, 2017. An Active Learning Approach for Improving the Accuracy of Automated Domain Model Extraction. *ACM Trans. Softw. Eng. Methodol.* V, N, Article A (January YYYY), 36 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Natural language (NL) is pervasive in requirements documents. It is estimated that ≈80% of what is typically known about a proposed system at the requirements analysis stage has been expressed in natural language [Luisa et al. 2004; Pohl 2010]. As software development progresses into specification and design, NL requirements inevitably need to be elaborated into more precise artifacts. An important artifact that

This work is supported by Luxembourg's National Research Fund (grant agreement number FNR/P10/03), and by the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement number 694277).
Author's addresses: C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, 29 Avenue J.F. Kennedy, L-1855 Luxembourg; emails: {arora, sabetzadeh, nejati, briand}@svv.lu;

DOI: http://dx.doi.org/10.1145/0000000.0000000

is commonly built during such elaboration is a *domain model* – an explicit representation of the salient concepts in an application domain and the relations between these concepts [Larman 2004].

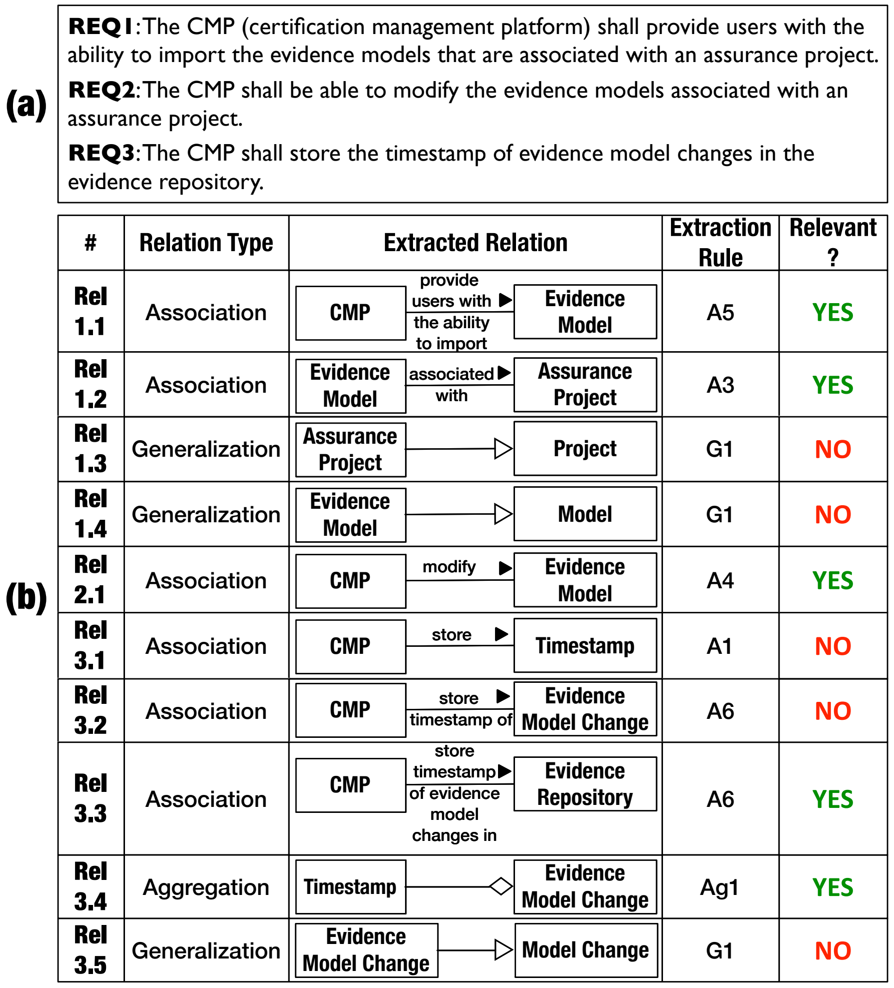
Domain models are valuable to the software development process in a number of ways: First, domain models provide structured knowledge about the terminology that underlies a domain. This makes domain models a useful instrument for communication between different stakeholders, including non-technical ones such as end-users [Larman 2004]. Second, domain models link a proposed system to the problem that the system is intended at addressing. In fact, the design of a system, particularly in a model-based development context, often takes shape around a domain model [Larman 2004]. Finally, domain models have gained considerable traction in recent years as an enabler for automated software verification. For example, model-based testing techniques rely heavily on domain models for automated test case and oracle generation [Utting and Legeard 2010; Wang et al. 2015].

Requirements documents may include hundreds and sometimes thousands of statements. Automated support presents a major advantage when one needs to build a domain model that is aligned with a large collection of NL requirements.

Numerous techniques exist for domain model extraction [Yue et al. 2011]. Most of these techniques use natural language processing (NLP) [Manning and Schütze 1999] as an enabling technology. Recent advances in NLP have made it possible to reliably extract increasingly more complex information from natural-language content. This has led to a surge of interest in developing new and improved solutions for automated requirements analysis, including automated domain model extraction [Arora et al. 2016; Thakur and Gupta 2016; Lucassen et al. 2017].

We illustrate domain model extraction through the example of Fig. 1. The NL requirements in Fig. 1(a) represent a small fragment of the requirements document for a safety certification platform. This platform is the subject of one of our case studies, as we explain later. To ease illustration, we have altered the requirements in Fig. 1(a) from their original form, without changing their substance. In Fig. 1(b), we show the distinct relations extracted from these requirements using the NLP-based domain model extractor developed in our earlier work [Arora et al. 2016]. The extracted relations are represented in the UML class diagram notation. The extraction rule behind each relation is further shown, e.g., A5 for Rel(ation) 1.1. We briefly describe in Section 2 our domain model extractor and the extraction rules that it implements.

The extracted relations are typically collated into a candidate domain model and subsequently presented to domain experts for review. Fig. 2(a) shows the candidate domain model induced by the relations of Fig. 1(b). As the experts review this model, they make decisions about what information should be retained in the domain model and what information should be filtered out. For example, for reasons that we discuss later in this section, only about half of the relations of Fig. 1(b) are retained and the remaining are discarded. In tandem or after this filtering process, the experts also decide about how they would like to represent the retained information. For example, the experts may elect to represent Rel 3.4 of Fig. 1(b) as an attribute, rather than an aggregation as in the candidate model. The associations may undergo changes as well. For example, associations Rel 1.1 and Rel 2.1 between "CMP" and "Evidence Model" may be merged into one relation with a more abstractly worded name, e.g., "managed in", attached to the merged relation. The experts may further introduce additional information that is absent from the candidate model. For instance, certain containment relations may have been left tacit in the requirements. For example, an "Evidence Repository" is part of an "Assurance Project", thus necessitating an aggregation from the latter to the former.



(a)

REQ1:The CMP (certification management platform) shall provide users with the ability to import the evidence models that are associated with an assurance project.

REQ2:The CMP shall be able to modify the evidence models associated with an assurance project.

REQ3:The CMP shall store the timestamp of evidence model changes in the evidence repository.

(b)

#	Relation Type	Extracted Relation	Extraction Rule	Relevant ?
Rel 1.1	Association	CMP — provide users with the ability to import ▶ — Evidence Model	A5	YES
Rel 1.2	Association	Evidence Model — associated with ▶ — Assurance Project	A3	YES
Rel 1.3	Generalization	Assurance Project —▷ Project	G1	NO
Rel 1.4	Generalization	Evidence Model —▷ Model	G1	NO
Rel 2.1	Association	CMP — modify ▶ — Evidence Model	A4	YES
Rel 3.1	Association	CMP — store ▶ — Timestamp	A1	NO
Rel 3.2	Association	CMP — store timestamp of ▶ — Evidence Model Change	A6	NO
Rel 3.3	Association	CMP — store timestamp of evidence model changes in ▶ — Evidence Repository	A6	YES
Rel 3.4	Aggregation	Timestamp —◇ Evidence Model Change	Ag1	YES
Rel 3.5	Generalization	Evidence Model Change —▷ Model Change	G1	NO

Fig. 1. (a) Example requirements; (b) Distinct extracted relations.

In Fig. 2(b), we show the outcome of the validation and elaboration process conducted by the experts over the automatically extracted candidate model. In the figure, we use colors to distinguish between the information that comes verbatim from the candidate model, the information that can be traced back to the candidate model but which has been altered from its original form, and the information that is absent from the candidate model and has been added manually by the experts. Ultimately, what the experts aim to obtain is a domain model that is feasibly complete [Lindland et al. 1994]: This means that accepting a model with any more or any less information would be less desirable than accepting the model as is. Naturally, the notion of completeness is contextual and dependent on how the experts wish to use the resulting domain model.

Challenge. Our work in this article concerns a challenge that we observed earlier in automated model extraction and already exemplified in Fig. 1 and Fig. 2. Upon presenting automatically extracted relations to a subject-matter expert in an industrial case study, the expert deemed only a fraction – about 36% – of the extracted relations *relevant*, i.e., useful for inclusion in the domain model. Despite the majority of the remaining relations being meaningful, the expert found them to be *superfluous*,

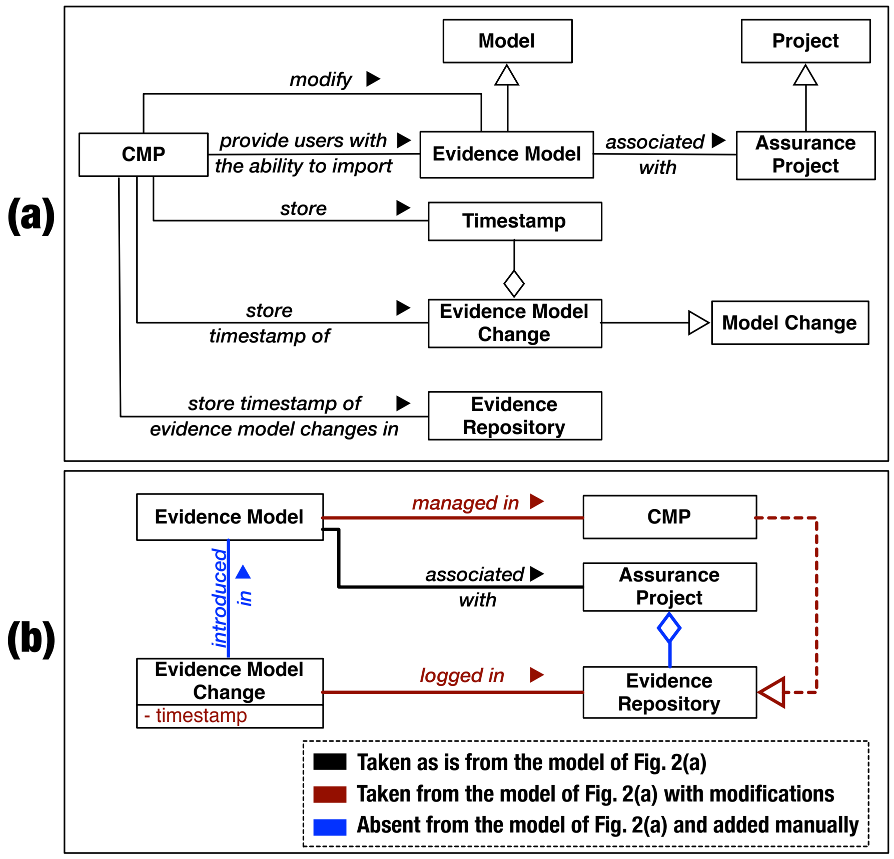


Fig. 2. (a) Automatically extracted domain model; (b) Domain model after being refined by an expert.

i.e., not pertinent to the domain model [Arora et al. 2016]. In the last column of the table in Fig. 1(b), we show the expert opinion about the relevance of the automatically extracted relations in our illustrative example. Various reasons were cited when a relation was found superfluous. For example, Rel 1.3, Rel 1.4 and Rel 3.5 were ruled out because these generalizations, including their target abstract concepts, namely “Project”, “Model” and “Model Change”, were found to be trivial and thus not useful. Rel 3.1 was too generic; and Rel 3.2 was discarded in favor of Rel 3.3.

The process that we outlined earlier, whereby the experts perform a completely manual review of the automatically extracted candidate model, is cumbersome. This is because the experts will have to manually filter numerous superfluous relations. If the experts filter too much, the resulting domain model will be of limited usefulness because it misses key domain knowledge. If the experts filter too little, again, the domain model will not serve the purposes it is built for, as the domain model is littered with superfluous information. Including superfluous information in the domain model is not only futile, but can also negatively affect understandability due to factors such as cognitive overload and clutter (e.g., when the domain model needs to be rendered visually).

As we argue more precisely in Section 3, by properly utilizing modern NLP technologies, one can automatically extract the large majority of relevant relations from the text of the requirements. This makes NLP a palatable basis for supporting the transition from NL requirements to domain models. At the same time, the ability to near-completely extract all the relevant relations comes at the cost of a considerable number of superfluous relations. Manually filtering the superfluous relations is still a better alternative than having to inspect all the requirements and manually extract all the relevant relations. Nevertheless, taking steps to reduce the number of superfluous relations (i.e., noise) without missing out on relevant relations would bring about major cost savings.

Objectives and Results. The main objectives of this article and the results achieved in relation to each objective are as follows.

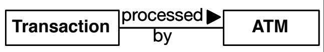
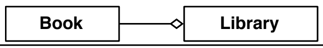
1) Building insights on relevance in model extraction: Our initial observation about relevance highlighted a little known issue in automated model extraction [Arora et al. 2016]. Our observation, however, was based on a single case study. Before deciding whether the observed problem warranted a technical solution, we needed to further examine whether the problem was representative of practice or an isolated case. To this end, we report on results from two additional studies, where we asked subject-matter experts about the relevance of domain model relations that were extracted automatically from industrial requirements. The results confirm our initial observation: relevant relations in the two new studies constitute only 39.5% and 36.6% of the extracted relations, respectively. In one of the new case studies, we had the opportunity to elicit feedback from multiple experts, and subsequently assess the level of agreement between them. Our interrator agreement analysis indicates that the experts are, to a large extent, consistent in how they reason about relevance. Nevertheless, we also found a non-negligible number of diverging opinions, suggesting that the experts exercise some degree of subjectivity.

2) Devising automated assistance for identifying superfluous relations: We propose an automated approach for assisting analysts in identifying superfluous relations while reviewing the domain model extraction results. Our approach builds on the concept of *active learning* – a machine learning paradigm in which a learning technique interactively requests inputs from an external source in order to improve the accuracy of the machine learning model [Settles and Craven 2008]. In our approach, we employ active learning to process analysts' feedback, and dynamically apply the logic gleaned from the feedback for reducing superfluous information. Specifically, we define a set of features for learning from analysts' feedback in the context of domain model extraction. Building on these features, we propose an algorithm for constructing a classifier that can assist analysts in identifying superfluous relations.

An effective realization of our automated assistance approach involves answering several questions. These include, among others, the choice of machine learning technique and the values to use for the approach's input parameters, e.g., the confidence margin necessary to ensure meaningful predictions. Using data from our three industrial case studies, we provide an optimal tuning of our approach. Subsequently, we examine the overall effectiveness of the approach over the case studies. We observe that, once trained, our approach automatically identifies an average of ≈45% of the superfluous relations with a precision of ≈96%. The results have the potential for major effort savings, noting that the recommendations are highly trustworthy and cover nearly half of the superfluous relations. The filtering rate achieved is particularly promising, considering the subjectivity that was observed among experts.

Structure. Section 2 provides background. Section 3 describes our studies on relevance. Sections 4 and 5 present our approach for filtering superfluous relations and our evaluation. Section 6 discusses threats to validity. Section 7 compares with related work. Section 8 concludes the article.

Table I. Domain model extraction rules.

Rule Id	Description	Example
C1	All noun phrases in the requirements are candidate concepts.	REQ1 in Fig. 1 :: **Certification Management Platform, CMP, User, Evidence Model**, and **Assurance Project**
A1	Transitive verbs suggest associations.	REQ3 in Fig. 1 :: Rel 3.1
A2	A verb with a preposition suggests an association.	"The transaction is processed by the ATM.":: 
A3	A relative clause modifier of nouns suggests an association.	REQ1 in Fig. 1 :: Rel 1.2
A4	A verbal clausal modifier suggests an association.	REQ2 in Fig. 1 :: Rel 2.1
A5	A non-finite verbal modifier suggests an association.	REQ1 in Fig. 1 :: Rel 1.1
A6	A prepositional dependency (Link Path) suggests an association between indirectly-related concepts.	REQ3 in Fig. 1 :: Rel 3.2
Ag1	Genitive cases suggest aggregations.	REQ3 in Fig. 1 :: Rel 3.4
Ag2	Terms "contain", "include", [...] suggest aggregations.	"The library contains books." :: 
At1	An intransitive verb with an adverb suggests an attribute.	"The train arrives in the morning at 10 AM." :: **Arrival time** is an attribute of **Train**.
G1	Premodifiers of noun phrases suggest generalizations.	REQ1 in Fig. 1 :: Rel 1.3

2. BACKGROUND

In this section, we first introduce domain model extraction. We then review the machine learning background for our approach.

2.1. Domain Model Extraction

To extract concepts and relations from NL statements, domain model extractors rely primarily on predefined rules that are implemented using NLP. This article is not concerned with the technicalities of model extraction: the extractor we draw upon has been discussed in our earlier work [Arora et al. 2016] and is publicly available at http://bit.ly/2nNNOTO. To be self-contained, we present and exemplify in Table I the main extraction rules that underlie our model extractor. Of the eleven rules in the table, one is for concepts (C1), six for associations (A1–A6), two for aggregations (Ag1–Ag2), one for attributes (At1), and one for generalizations (G1). Our model extractor further has rules for assigning multiplicities to associations. We do not discuss these rules here. For details, see [Arora et al. 2016]. Rules A3 to A6 are specific to our domain model extractor; the remaining rules are common and used by several other extractors as well [Yue et al. 2011; Elbendak et al. 2011; VidyaSagar and Abirami 2014; Thakur and Gupta 2016; Lucassen et al. 2017].

Among the rules in Table I, A6 requires some further explanation. Rather than just one rule, A6 represents a class of rules known as *Link Paths (LP)* [Akbik and Broß 2009; Fader et al. 2011]. As opposed to rules A1 – A5 in Table I which extract *direct* associations between the concepts in a sentence, rule A6 extracts *indirect* associations. To illustrate the notions of direct and indirect, consider Rel 3.1, Rel 3.2 and Rel 3.3 in the example of Fig. 1. The grammatical dependencies that induce these associations

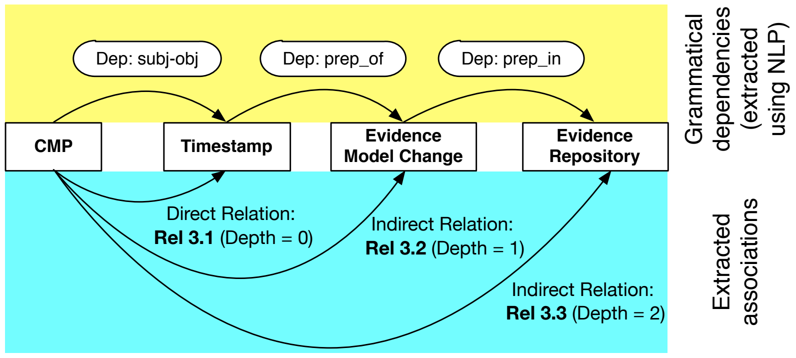

Dep: subj-obj
Dep: prep_of
Dep: prep_in
CMP
Timestamp
Evidence Model Change
Evidence Repository
Grammatical dependencies (extracted using NLP)
Direct Relation: Rel 3.1 (Depth = 0)
Indirect Relation: Rel 3.2 (Depth = 1)
Indirect Relation: Rel 3.3 (Depth = 2)
Extracted associations

Fig. 3. Illustration of Link Paths over REQ3 from the example of Fig. 1.

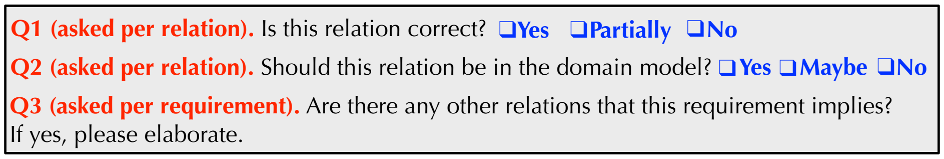
are shown in Fig. 3. Here, Rel 3.1 is a direct association because it is derived from the subject-object dependency between *CMP* and *Timestamp*. In contrast, Rel 3.2 and Rel 3.3 are indirect, as they are deduced from additional dependencies – in this case prepositional dependencies – originating from the target concept of Rel 3.1 (i.e., *Timestamp*). The prepositional dependency between *Timestamp* and *Evidence Model Change* induces Rel 3.2. This dependency, when chained to the prepositional dependency between *Evidence Model Change* and *Evidence Repository*, induces Rel 3.3. The number of additional dependencies that are chained to a direct association denotes the *depth* of an LP association. For example, Rel 3.2 has an LP depth of *one*, whereas Rel 3.3 has an LP depth of *two*. Direct associations such as Rel 3.1, which serve as base relations for LP associations, have a depth of *zero*. Our model extractor identifies LP associations up to a maximum depth of four.

We use the term *LP group* to refer to a direct association alongside all its indirect associations extracted using rule A6. For example, the set $\{\text{Rel } 3.1, \text{Rel } 3.2, \text{Rel } 3.3\}$ is an LP group. As we elaborate in Section 3, we empirically observe an interesting property that facilitates the filtering of superfluous associations in LP groups. Our technical solution in Section 4 utilizes a combination of this property and machine learning for accurate identification of superfluous relations.

2.2. Using Machine Learning (ML) for Building Recommendation Systems

Using ML is common in software engineering for building recommendation systems. Recommendation systems are “software applications that provide information items estimated to be valuable for software engineering tasks in a given context” [Robillard et al. 2014]. Our work relates most closely to *Recommendation systems In-The-Small (RITS)*, where recommendations are drawn based on a small amount of data taken from the analyst’s immediate context [Robillard et al. 2014]. RITS are commonly implemented using ML so that the logic and rationale behind the recommendations can evolve as more data and analyst feedback become available [Robillard et al. 2014]. This is particularly important in our context, where recommendations have to adapt on-the-fly to different requirements documents in different application domains.

Our approach is a *supervised* method: it requires analysts to label relevant and superfluous items, and then uses these labels for training. Supervised learning techniques are divided into *regression* and *classification*, where the goal is to predict real-valued and categorical outputs, respectively [Louridas and Ebert 2016]. Our approach falls under classification, since its function is to tell apart the relevant and superfluous categories. In our evaluation (Section 5), we experiment with several classification techniques in order to determine which one is most accurate in our context.



Q1 (asked per relation). Is this relation correct? ❑Yes ❑Partially ❑No

Q2 (asked per relation). Should this relation be in the domain model? ❑Yes ❑Maybe ❑No

Q3 (asked per requirement). Are there any other relations that this requirement implies? If yes, please elaborate.

Fig. 4. Interview survey questionnaire.

Our technical solution builds upon *active learning*. An active learning technique starts with a seed training set and iteratively samples the unlabeled data set for further learning [Settles 2012]. Active learners are typically instantiated based on a sampling strategy that is aimed at maximizing the effectiveness of learning [Yu et al. 2018]. Notable sampling strategies include selecting (unlabeled) data over which the learner is most certain [Miwa et al. 2014] or least certain [Lewis and Gale 1994], data that is likely to change the current learned model the most [Settles and Craven 2008], and data that is most representative of the unlabeled data set [Settles and Craven 2008; DeBarr and Wechsler 2009]. In our approach, we sample the most uncertain data. This sampling approach is one of the most common, and is often referred to as Simple Active Learning (SAL) [Cormack and Grossman 2014]. We discuss our algorithm and the way it implements SAL in Section 4.

3. EXPERT SURVEYS ON RELEVANCE

This section presents the design, execution, and results of three interview surveys with subject-matter experts on the output that our domain model extractor generates over industrial requirements. The overall survey protocol as well as the results for one of the three surveys comes from our previous work [Arora et al. 2016]. For completeness, we summarize from this past work what is needed for this article. In addition to providing further insights about relevance in domain model extraction, the data collected in the surveys serves as the gold standard for evaluating the approach we propose in Section 4.

3.1. Survey Design and Execution

Our survey was designed to elicit domain experts' responses to the questionnaire of Fig. 4. Of the three questions, Q1 to Q3, on the questionnaire, we are interested, for the purposes of this article, in Q2 only. Nevertheless, to be able to properly interpret the results of Q2, we need the results of Q1 and Q3 as well.

Before conducting the surveys, we explained and illustrated to the experts what we meant by the term "relation" in the questions. Specifically, we defined a relation to be a *(regular) association*, an *aggregation*, a *generalization*, or an *attribute*. We treat attributes as relations, because of the "belongs to" link between an attribute and its concept. The questionnaire does not directly address the extracted concepts. The rationale is that concepts appear as the endpoints of the relations; concepts are therefore always reviewed as part of the relations being examined.

3.1.1. Survey Material. Table II provides information about the material used in our three surveys, denoted Case A, Case B and Case C. The source requirements in all three cases were collections of (IEEE-830-style) "shall" statements. Case A is from our previous work; Case B and Case C are new. In each survey, we aimed to cover at least 30% of the requirements. The requirements to cover were picked randomly. We could not cover all the requirements due to the limited availability of the survey participants (domain experts). Table II shows the total number of requirements in each case, the

Table II. Content used in interview surveys.

Case	Document Description	# of Requirements	# of Requirements Covered in Survey	# of Distinct Relations Examined by Experts
Case A	Requirements for a satellite system simulator	158	50 (31.6%)	213
Case B	Requirements for a safety information management system	110	33 (30%)	157
Case C	Requirements for an automotive occupant classification system	79	27 (34.2%)	101

number of requirements covered in the survey, and the number of (distinct) relations induced by the covered requirements.

3.1.2. Survey Participants. In each Case A and Case B, an individual expert provided feedback on the relations extracted from the selected requirements (two distinct individuals for the two case studies). In both cases, the expert involved was the lead author and analyst of the requirements under investigation. In Case C, three experts agreed to participate in our survey. These three were: the lead requirements author (also, the project manager) and two additional domain experts who worked as design and development engineers. As we describe below, the surveys were organized into sessions. In Case C, two sessions were held, the first of which was attended by all the three experts. Due to unforeseen circumstances, however, only the lead requirements author in Case C could participate in the second session. Throughout the entire article, we use for Case C only the feedback obtained from this single expert. The only exception is in Section 3.2.4, where we use the (incomplete) feedback from the first session of Case C for analyzing the degree of agreement among the three participating experts. We note that the experts involved in all Case A, Case B, and Case C were experienced in domain modeling and fully familiar with the requirements from which the relations had been extracted.

3.1.3. Survey Execution. In each case, the expert was asked to examine, through Q1 and Q2, the individual relations extracted from a specific requirement. After all the relations extracted from a given requirement REQ had been evaluated, the expert was asked, through Q3, whether she could think of additional relations that were implied by REQ, but which were not present in the extracted results. The relations extracted from each requirement were presented to the expert in the same visual representation as shown by the third column of Fig. 1(b).

Q1 concerns *correctness*. For a given relation, the expert was instructed to answer *Yes*, if she deemed all the following conditions to be met simultaneously: (1) the concept (or attribute) at each endpoint of the relation is meaningful, (2) the relation itself is meaningful with the correct type assigned to it (e.g., generalization or association), and (3) if the relation happens to be an association, the relation has the correct name and multiplicity constraint assigned. We instructed the expert to respond by *Partially* when she identified any omission or inaccuracy with respect to the conditions above, but she found the inaccuracy or omission not to be major and thus not affecting the meaningfulness of the relation. The expert was otherwise asked to answer Q1 by *No*.

Q2 addresses *relevance* – our main topic of investigation in this article. As stated before, relevance has to do with whether experts deem a relation useful for inclusion in the domain model. For a given relation, the expert was asked Q2 only if they had answered *Yes* or *Partially* to Q1. A *No* answer to Q1 prompted an automatic *No* answer to Q2. If the expert had responded to Q1 by *Partially*, we asked them to explain any omissions / inaccuracies observed. We recorded (wrote down) the explanation given by the expert. Subsequently, we asked the expert to answer Q2 based on the premise

Table III. Interview survey results.

Case	% of Partially Correct and Correct Relations (Q1 Results)	% of Relevant + Maybe Relevant Relations (Q2 Results)	# of Missing Relations (Q3 Results)	% of Relevant Relations Retrieved *
Case A	191/213 *(89.7%)*	(71+5)/213 *(35.7%)*	7	91.6%
Case B	135/157 *(86.0%)*	(60+2)/157 *(39.5%)*	8	88.6%
Case C	86/101 *(85.1%)*	(37+0)/101 *(36.6%)*	5	88.1%

$$\text{* relevant relations retrieved} = \frac{\text{\# of relevant (incl. maybe) relations}}{\text{\# of relevant (incl. maybe) relations + \# missing relations}}$$

that the omissions / inaccuracies have been addressed. A *Maybe* option was provided for Q2 to handle situations where the expert could not readily decide about relevance. Whenever the experts' response to Q2 was *Maybe* or *No*, the experts were further asked to explain their rationale. The explanation, which was spontaneous and unguided, was scribed by two researchers (first two authors), and later cross-checked and reconciled.

Lastly, Q3 asks about relations that have been *missed* by automated extraction, but which the expert could identify upon a manual examination of a given requirement. The experts answered Q3 only after having gone through *all* the relations that were automatically extracted from a given requirement.

The interview for Case A took ≈6 hours (three two-hour sessions on different days). The interviews for Case B and Case C took ≈4 hours each (two two-hour sessions on different days). In Case C, as noted before, three experts participated in the first session and only one expert was available for the second session.

3.2. Survey Outcomes

In this section, we present the results from our interview surveys and the insights gained from analyzing these results.

3.2.1. Results. Table III shows the overall survey results. The results for Q1 indicate that the large majority ($> 85\%$) of the extracted relations are correct, either partially or fully. This provides evidence for the meaningfulness of the extracted relations via NLP-based extraction rules. With regard to Q2, the highest relevance ratio observed is 39.5% (Case B). To ensure that low relevance is not an issue that applies exclusively to our model extractor, we break down the relevance results per extraction rule. This rule-wise breakdown, shown in Table IV, indicates that the new extraction rules in our model extractor –A3 to A6 in Table I– are not contributing disproportionately to low relevance when compared to the other rules in Table I which are common across other existing tools. This suggests that low relevance is a general problem and *not* restricted to our model extraction solution per se.

In light of our results, we conclude that an explicit solution for improving relevance would be necessary. Developing such a solution would of course make sense only if one can already extract the large majority of relevant relations via NLP. This is addressed by Q3. In particular, our results in Table III indicate that the number of missed relations is proportionally low, with more than 88% of the relevant relations being retrievable via NLP in our studies. Consequently, one can expect useful gains from automated assistance for distinguishing relevant and superfluous relations.

Our analysis of the survey results led to a number of additional findings about relevance in domain model extraction. We outline these findings in the rest of this section.

3.2.2. Reasons for superfluousness. As per our survey design, incorrect relations were automatically marked as superfluous. With the incorrect relations excluded, the ex-

Table IV. Rule-wise breakdown of relevance results.

	(Relation) Extraction Rule									
	A1	A2	A3	A4	A5	A6	Ag1	Ag2	At1	G1
Case A (Relevance)										
Y	12	1	3	7	7	26	8	0	0	7
M	0	0	0	0	1	0	0	0	0	4
N	17	3	4	13	7	48	14	17	2	12
Case B (Relevance)										
Y	9	2	2	1	6	18	15	3	4	0
M	0	0	0	0	0	2	0	0	0	0
N	6	1	4	2	7	50	12	0	1	12
Case C (Relevance)										
Y	2	8	2	1	1	14	5	2	0	2
M	0	0	0	0	0	0	0	0	0	0
N	7	5	3	2	1	19	6	1	0	20
Overall Relevance per Extraction Rule										
Y	23	11	7	9	14	58	28	5	4	9
M	0	0	0	0	1	2	0	0	0	4
N	30	9	11	17	15	117	32	18	3	44
R%	43.4%	55.0%	38.9%	34.6%	50.0%	33.9%	46.7%	21.7%	57.1%	22.8%

Legend **Relevance:** **Y** Yes **M** Maybe **N** No **R%** Relevance (%)

perts answered Q2 for 191 relations in Case A, 135 relations in Case B, and 86 relations in Case C. From these relations, the experts deemed superfluous 115 relations in Case A, 73 relations in Case B, and 49 relations in Case C. For a small number of relations – five in Case A and two in Case B – the experts were uncertain as to whether the relations should be included in the domain model, hence answering Q2 by *maybe*.

In Table V, we provide a classification of the rationale given by the experts when they found a relation to be superfluous, or when they could not give a conclusive answer about relevance. This classification was developed after the conclusion of the interview surveys, and following established guidelines for qualitative analysis [Auerbach and Silverstein 2003]. Specifically, we derived labels for the rationale items using words and phrases from the experts' terminology in the interview surveys. This is known as in-vivo coding [Saldaña 2015]. The labels were iteratively merged and abstracted into themes. The themes were then reviewed and finalized through discussion among the researchers. Due to the chronological order in which our studies were performed, the themes were first developed for Case A and later used as reference in Case B and Case C. The results of our qualitative analysis were then validated by the experts, who agreed that the classification of Table V was an accurate characterization of the reasons for (potential and actual) superfluousness.

Table V further shows for each category (theme) in our classification the number of relations falling under that category as well as an example. The first row of the table, "future contingencies", captures situations where the experts were unsure about the relevance of a given relation, because the decision depended on future events and circumstances. The second row, "too detailed", captures situations where the experts considered the information conveyed by a relation to be too specific for the domain model. The third row, "too unspecific", is the opposite, i.e., the information was too abstract or lacking sufficient context. The final row, "trivial knowledge", captures situations where the experts considered a relation to be too obvious and widely known by the stakeholders, and thus not warranted in the domain model.

3.2.3. Relevance of associations in Link Path (LP) groups. Our analysis revealed an important and, in retrospect, intuitive trend concerning the relevance of associations in LP groups (the notion of LP group was defined in Section 2.1). Specifically, we observed

Table V. Reasons for superfluousness.

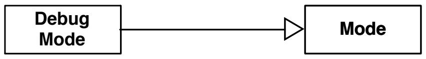
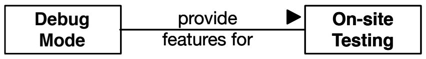
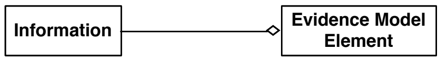
	#	Reason	Case A Count	Case B Count	Case C Count	Example
Maybe Relevant	1	Future Contingencies	5	2	0	“The CMP shall support a debug mode, which provides features for on-site testing.”  **Expert Feedback:** *Currently there is only one mode — debug mode — for the system. This may change, in which case this relation will be relevant.*
Superfluous	2	Relation Too Detailed	88	53	23	“The CMP shall support a debug mode, which provides features for on-site testing.” 
	3	Relations Too Unspecific / Lacking Context Information	21	17	10	Rel 3.1 and Rel 3.2 in Fig. 1
	4	Trivial Knowledge	6	3	16	“The CMP shall maintain an evidence repository with the information of evidence model elements.” 

Table VI. Statistics about LP groups and the number of relevant associations in them.

Case	# of LP groups	Avg. # of associations in LP group	# of LP groups with at most one relevant association	# of LP groups with two or more relevant associations
Case A	38	2.95	36	2
Case B	32	3.19	30	2
Case C	19	2.74	18	1

that once an association from an LP group L was deemed relevant by an expert, the expert was unlikely to find another association in L relevant. Table VI provides statistics about the LP groups in our case studies. For each case study, the table shows the number of LP groups, the average number of associations in the groups, the number of groups with zero or one relevant association, and the number of groups with more than one relevant relation. Quantitatively speaking and across our three case studies, for $(36 + 30 + 18)/(38 + 32 + 19) \approx 94\%$ of the LP groups, the experts found *at most one* association to be relevant. We note that the experts in our case studies had no knowledge of the rules employed by our model extractor, nor the fact that some associations were related to one another through some underlying scheme (Link Paths).

The above phenomenon can be explained as follows: LP groups are manifestations of the same relation with different amounts of contextual information (constraints) embedded in them [Fader et al. 2011]. What we observed in our case studies was that the experts – without being aware of the existence of such notion as LP groups – sought relations with the optimal level of contextual information in them. If the relation with the desired level of detail happened to be from an LP group, the remaining relations in the group were naturally found to be either too unspecific or too detailed.

To further analyze the properties of LP groups, we provide in Fig. 5 a depth-wise breakdown of relevant associations in these groups. As seen from the figure, the majority of relevant associations in LP groups have a depth of zero (direct association) or one (an additional concept chained to the direct association). Associations with a depth of two are not uncommon, although they are less prevalent than those with a

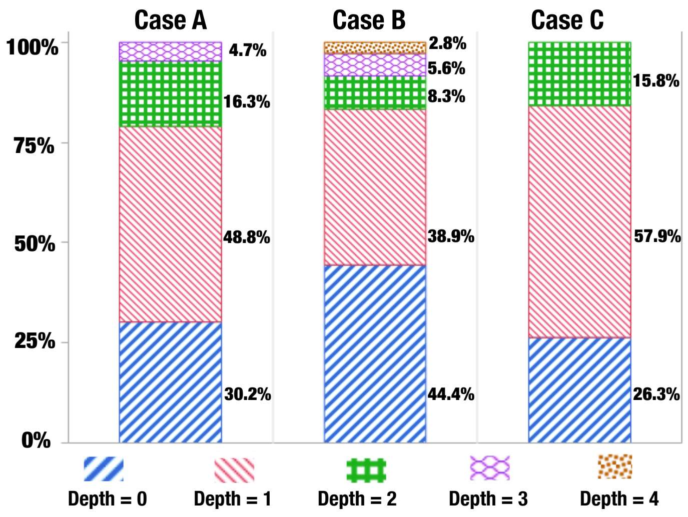


Fig. 5. Depth-wise breakdown of relevant associations in LP groups.

depth of zero or one. Associations with depths of three and higher are relatively uncommon, since these associations require cascades of three or more prepositional phrases. Such cascades occur predominantly in long sentences, which requirements writing best practices advise against [Génova et al. 2013]. Furthermore, higher-depth associations, should they be present, are unlikely to have much practical value, because they establish relationships between concepts that are too far apart. As noted in Section 2.1, our model extractor finds associations with a maximum LP depth of four.

We conclude our discussion of LP groups with a final remark about the way we utilize these groups in Section 4 for filtering superfluous relations. Due to multiple relevant relations from the same LP group being unlikely to occur (probability of $< 6\%$), we incorporate a rule into our approach to filter all the remaining associations of an LP group, once some association from the group has been deemed relevant.

3.2.4. Subjectivity in decisions about relevance. As mentioned above, for part of Case C we had access to three different experts. Specifically, for 50 out of the total of 101 relations in Case C, we collected feedback from all three experts. We employed a simple round-table meeting for eliciting expert opinions, with each expert responding to our questionnaire in the presence of other experts. The experts were in perfect agreement over their answers to Q1, and unanimously found 46 of the relations to be correct or partially correct, and the remaining four incorrect.

The experts answered Q2 for the 46 correct and partially correct relations. In contrast to Q1, the experts had diverging opinions on 13 relations. To quantify the level of agreement between the experts, we apply Fleiss' Kappa (κ) [Fleiss 1971]. For the responses to Q2, we obtain $\kappa = 0.65$. This κ score denotes *substantial agreement* [Landis and Koch 1977], thus providing evidence that the experts applied a largely consistent reasoning process when determining relevance.

Of the 13 disagreements between the experts, ten were rooted in the level of detail the experts wished to see in the relations. In a post-study analysis, we observed that nine of these ten relations originated from LP groups. For the remaining three out of the 13 relations, the disagreement was over whether the relations were worthwhile or too trivial for being modeled.

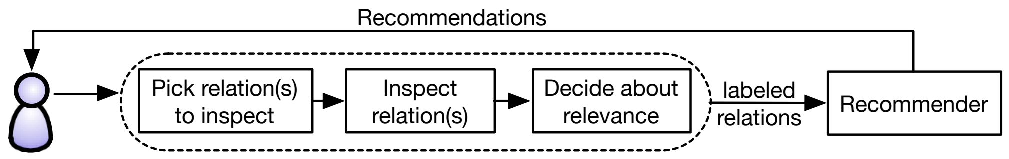


Fig. 6. Approach overview.

The disagreements suggest that there is a degree of subjectivity involved in telling apart relevant relations from superfluous ones. Such subjectivity has indeed been already observed in other contexts, e.g., web mining [Kosala and Blockeel 2000] and media report analysis [Scholz and Conrad 2013], when experts were tasked with identifying relevant information from the output of NLP technologies. While subjectivity inevitably reduces the accuracy of automation, the level of interrator agreement observed in our context (i.e., domain model extraction) leaves ample room for building useful automated assistance for filtering superfluous information.

4. APPROACH FOR FILTERING SUPERFLUOUS RELATIONS

Fig. 6 illustrates the idea behind our automated assistance approach: given a set of relations, we build a recommender that predicts based on a classifier a label – relevant or superfluous – for each relation. These predictions come with probabilities that indicate prediction confidence. The labeled relations and the probabilities can then be presented to the analyst in a visually-enhanced form, e.g., via a sorted list or a color-coding scheme where different probability ranges are rendered in different colors. Using such cues, the analyst can pick one or more relations, inspect them, and either accept or reject the predicted labels. The updated labels are subsequently used to reconstruct a more accurate classification model for the remaining relations that are yet to be inspected by the analyst.

The effectiveness of such a recommender would naturally be affected by which relations the analyst picks to inspect in each iteration. In lieu of having a human analyst in the loop, we provide an instantiation of the approach whereby the next relations to inspect are picked by an automated strategy. This strategy, alongside the expert survey data collected in Section 3, enable us to *simulate* the analyst's input to the recommender. By so doing, as we elaborate in Section 5, we are able to make a number of key technical decisions that are necessary prerequisites to running future experiments with human subjects.

Specifically, our strategy in regard to the next relations to show to the analyst for inspection is as follows: if, in a given iteration, a relation is predicted to be relevant with high probability, we accept the prediction and *recommend* it to the analyst. In contrast, we do not accept a prediction of being superfluous the first time it is made, no matter how high the probability is. Rather, we wait for multiple iterations and monitor whether the prediction remains the same. Stated otherwise, our recommender withholds a recommendation of "superfluous" for multiple rounds in order to build confidence that the predictions remain stable with more feedback from the analyst becoming available.

Our asymmetric treatment of predictions for relevance and superfluousness is motivated by two reasons: (1) we need to strictly avoid misleading the analyst toward filtering useful relations, and (2) to save manual effort, we are interested primarily in accurately identifying superfluous relations, rather than the relevant ones. This is because a recommendation of "relevant" conveys no information about the correctness of a relation (as defined in Section 3). Relevant relations are thus always subject to a careful manual analysis to ensure that all their details are correct. This is in con-

```
Algorithm RECOMMENDER
Input:        - Set $R$ of relations extracted from requirements
              - Feature functions $f_1, \ldots, f_l$
Parameters:   - Training set size $n$
              - Confidence margins $\epsilon$ and $\delta$
              - Iteration parameter $k$
Output:       - Recommendations of relevance and superfluousness

1.   Pick a (random) training set $T \subseteq R$ such that $|T| = n$ and let $X = R \setminus T$
2*.  Obtain analyst feedback on every relation $r \in T$
3.   Split $T$ into $T^+$ and $T^-$ based on the analyst feedback
     /* $T^+$ contains relevant and $T^-$ contains superfluous relations. */
4.   Build $FeatureMatrix$ based on $f_1(T), \ldots, f_l(T)$
5.   Build classifier $\mathcal{C}$ over $FeatureMatrix$
6.   repeat /* Active learning loop */
7.     Apply $\mathcal{C}$ to $X$, and for every $r \in X$:
       let $p(r)$ be the prediction probability of $r$ being superfluous
8.     $L_{min} = \min\{p(r)\}_{r \in X}$
9.     $L_{max} = \max\{p(r)\}_{r \in X}$
10.    $I^+ = \{r \in X \mid p(r) \leq L_{min} + \delta\}$
11.    $I^- = \{r \in X \mid (p(r) \geq L_{max} - \epsilon) \wedge ($r has been predicted with high probability to be superfluous
                                                      in $k-1$ of the previous iterations$)\}$
12.    $I^- = I^- \cup \{r \in X \mid$ there is some $r' \in T^+$ s.t. $r$ and $r'$ are in the same LP group$\}$
       /* See the discussion in Section 3.2.3. Once a relation from an LP group lands in $T^+$,
          we recommend as superfluous all other relations from that LP group. */
13.    If $I^+ \cup I^- \neq \emptyset$ then
14.      $X = X \setminus (I^+ \cup I^-)$
15*.     Present $I^+$ and $I^-$ to the analyst; the analyst checks that elements in $I^+$ (resp. $I^-$) are indeed
         relevant (resp. superfluous); analyst modifies $I^+$ and $I^-$ as necessary
16.      $T^+ = T^+ \cup I^+$
17.      $T^- = T^- \cup I^-$
18.    else
19*.     Let $r \in X$ be a relation for which there is minimum support of superfluousness;
         obtain analyst feedback on $r$
20.      Add $r$ to $T^+$ if relevant and to $T^-$ if superfluous
21.      $X = X \setminus \{r\}$
22.    Update $FeatureMatrix$ based on $f_1(T^+ \cup T^-), \ldots, f_l(T^+ \cup T^-)$
23.    Rebuild $\mathcal{C}$ over $FeatureMatrix$
24.  until $X = \emptyset$
```

Fig. 7. Algorithm for making recommendations on relevance and superfluousness. The lines labeled with an asterisk (*) require user interaction.

trast to superfluous relations, which, once identified, can be immediately dismissed. In our evaluation of Section 5, we focus exclusively on our approach's ability to identify *superfluous* relations.

If, in a given round, the recommender cannot issue any recommendation, we still need to present the analyst with a relation for feedback so that the process can continue. In such a case, we attempt to pick a relation that has a low likelihood of receiving an automated recommendation in the future.

The algorithm of Fig. 7, named RECOMMENDER, presents our approach more precisely. The algorithm receives as input a set R of relations extracted from natural-language requirements and feature functions $f_1, \ldots, f_l$. The values of the feature functions are used for building and training a classifier $\mathcal{C}$. Our feature functions are listed in Table VII. For each feature in the table, we provide an id, a definition, and the intuition captured by the feature alongside an example. The algorithm further has four parameters, n, ϵ, δ and k, which are used for tuning the learning process. Suitable values for these parameters are derived empirically in Section 5.

Table VII. Features for learning.

	Id	Definition	Intuition and *Example*
Label-independent Features	**IF1**	Relation type (Association, Aggregation, Attribute, Generalization).	Type is an inherent characteristic of a relation and could thus be a useful indicator for building a machine learning model. ***Rel 2.1 in Fig. 1 :: IF1 = A (Association)***
	IF2	The rule that extracted the relation (see Table I).	Similar to the intuition for IF1. ***Rel 2.1 in Fig. 1 :: IF2 = A4***
	IF3	If the relation is part of an LP group, IF3 is the LP depth of the relation (see Fig. 3). Otherwise (i.e., when the relation does not belong to an LP group), IF3 is -1, indicating that IF3 does not apply.	The depth of an LP association is the number of additional concepts that participate in indirectly linking a given pair of concepts. Based on our empirical observations (Section 3.2.3), LP relations with high depths are unlikely to be relevant. Subsequently, IF3 may be useful for relevance classification. ***Rel 3.1 in Fig. 1 :: IF3 = 0 (direct association with extended LP associations - Rel 3.2 and Rel 3.3)*** ***Rel 3.2 in Fig. 1 :: IF3 = 1 (one added concept "Timestamp")*** ***Rel 3.4 in Fig. 1 :: IF3 = -1 (not applicable)***
	IF4	Ratio of the cumulative length of all tokens in the relation (source concept, target concept and relation name in the case of associations) to the length of all tokens in the requirement from which the relation has been extracted.	A relation incorporating a large text segment of a given requirement is unlikely to be useful, since one may as well read the original requirement. ***Rel 2.1 in Fig. 1 :: IF4 = (# of tokens in Rel 2.1) / (# of tokens in R2) = 4 / 15 = 0.27***
	IF5[1] **IF6**[1]	{Minimum (IF5), Maximum(IF6)} of the number of tokens in the source concept and the target concept of a relation.	Concept names that are too long or too short could be an indication of unwanted concepts. ***Rel 2.1 in Fig. 1 :: IF5 = min(1,2) = 1 (min(# of tokens in "CMP", # of tokens in "Evidence Model"))*** ***Rel 2.1 in Fig. 1 :: IF6 = max(1,2) = 2***
	IF7	If any of the end concepts in a relation is a single-token word which is defined in an English dictionary.	Single-token words are often not useful as domain concepts. ***Rel 1.4 in Fig. 1 :: IF7 = TRUE***
	IF8	If any of the end concepts in a relation contains a conjunction, such as "and".	If an end concept of a relation has a conjunction in it, the end concept and thus the relation may have a larger likelihood of being incorrect. ***If in Rel 3.1, "Timestamp" is replaced by "Time and Date" :: IF8 = TRUE***
Label-dependent Features[2]	**DF1**	The number of **relevant** relations in the training set that have been extracted from the same requirement and via the same extraction rule as the relation in question.	The relevance of other relations extracted under similar conditions may be a useful indicator for relevance. ***Rel 2.1 in Fig. 1 :: DF1 = 0 (No other relation in the training set is extracted from R2 using rule A4)***
	DF2[1] **DF3**[1]	{Minimum (DF2), Maximum(DF3)} of the following two numbers: (a) the number of **relevant** relations in the training set that have as an endpoint the source concept of the relation in question (b) the number of **relevant** relations in the training set that have as an endpoint the target concept of the relation in question.	If the concepts at the two endpoints of the relation have appeared in other relevant relations, it may be more likely for the relation in question to be relevant. ***Rel 2.1 in Fig. 1 :: DF2 = min(1,2) = 1 (One relevant relation, Rel 1.1, shares the "CMP" end concept, and two relevant relations, Rel 1.1 and Rel 1.2 share the "Evidence Model" end concept)*** ***Rel 2.1 in Fig. 1 :: DF3 = max(1,2) = 2***

Label-dependent Features[2]	**DF4**	For a given relation *r*, let *S(r)* denote the union of all the noun phrases and verbs appearing in *r*'s endpoints and *r*'s name (in case of associations). Let *q* be the relation in question. DF4 is the number of **relevant** relations *r* in the training set where $S(r) \subseteq S(q)$.	The intuition is similar to that for DF2 and DF3. The distinction is that DF4 simultaneously considers *all* the textual components of relevant relations, including relation names (in case of associations). ***Rel 3.2 in Fig. 1 :: DF4 = 1 (All textual components of Rel 3.4 -a relevant relation- are contained in Rel 3.2)***
	DF5	The number of **relevant** relations in the training set that share with the relation in question both of their endpoints.	Any relation satisfying the condition of DF5 would go in parallel to the relation in question. Parallel relations make sense only between certain concepts. By keeping track of the relevance of parallel relations, DF5 can be a useful indicator for predicting the relevance of relations that go in parallel to those already in the training set. ***Rel 2.1 in Fig. 1 :: DF5 = 1 (Only one relevant relation in the training set, Rel 1.1, shares both the end concepts of Rel 2.1)***
	DF6	The number of **relevant** relations in the training set that share the same verb as the relation in question (for associations only).	A verb shared with a relevant association can, alongside other factors such as IF3, serve as an indicator for relevance. ***Rel 3.2 in Fig. 1 :: DF6 = 0 (Only one relation in the training set, Rel 3.1, shares the verb "store" with Rel 3.2; however Rel 3.1 is superfluous)***
	DF7	Replace 'relevant' in DF1 with '**superfluous**'.	Same intuition as that for DF1, replace 'relevant' with 'superfluous'. ***Rel 2.1 in Fig. 1 :: DF7 = 0 (see explanation of DF1)***
	DF8[1] **DF9**[1]	Replace 'relevant' in DF2 and DF3, respectively, with '**superfluous**'.	Same intuition as that for DF2 and DF3; replace 'relevant' with 'superfluous'. ***Rel 2.1 in Fig. 1 :: DF8 = min(1,2) = 1 (One superfluous relation, Rel 1.4 shares the "Evidence Model" end concept, and two superfluous relations, Rel 3.1 and Rel 3.2, share the "CMP" end concept)*** ***Rel 2.1 in Fig. 1 :: DF9 = max(1,2) = 2***
	DF10	Replace 'relevant' in DF4 with '**superfluous**'.	Same intuition as that for DF4; replace 'relevant' with 'superfluous'. ***Rel 3.2 in Fig. 1 :: DF10 = 1 (Rel 3.1 is a subset of Rel 3.2)***
	DF11	Replace 'relevant' in DF5 with '**superfluous**'.	Same intuition as that for DF5; replace 'relevant' with 'superfluous'. ***Rel 2.1 in Fig. 1 :: DF11 = 0 (see explanation of DF5)***
	DF12	Replace 'relevant' in DF6 with '**superfluous**'.	Same intuition as that for DF6; replace 'relevant' with 'superfluous'. ***Rel 2.1 in Fig. 1 :: DF12 = 1 (see explanation of DF6)***
	DF13	For a given relation *r*, let *S(r)* denote the union of all the noun phrases and verbs appearing in *r*'s endpoints and *r*'s name (in case of associations). Let *q* be the relation in question. DF13 is the number of **relevant** relations *r* in the training set where $S(r) \supseteq S(q)$.	The textual components of a relation, i.e., the endpoints and the relation names (in case of associations) can be useful for determining relevance. ***Rel 3.1 in Fig. 1 :: DF13 = 0 (Rel 3.2 shares all the concepts and verbs in Rel 3.1, however it is superfluous)***
	DF14	Replace 'relevant' in DF13 with '**superfluous**'.	Same intuition as that for DF13; replace 'relevant' with 'superfluous'. ***Rel 3.1 in Fig. 1 :: DF14 = 1 (Rel 3.2 shares all the concepts and verbs in Rel 3.1)***
	DF15	The number of **relevant** relations in the training set that have been extracted from the same requirement as the relation in question.	The relevance of other relations extracted from the same requirement may be a useful indicator for relevance. ***Rel 1.1 in Fig. 1 :: DF15 = 1 (Rel 1.2 is relevant and has been extracted from the same requirement as Rel1.1)***
	DF16	Replace 'relevant' in DF15 with '**superfluous**'.	Same intuition as that for DF15; replace 'relevant' with 'superfluous'. ***Rel 1.1 in Fig. 1 :: DF15 = 2 (Rel 1.3 and Rel1.4 are superfluous and have been extracted from the same requirement as Rel1.1)***

[1] *The minimum and maximum operators enable the treatment of source and target concepts in a symmetric way, without having to define separate features for each endpoint of a relation.*

[2] *The examples for label-dependent features are based on the* matrix *in Fig. 8*.

The algorithm starts by partitioning R into sets T and X (line 1), with the number of relations in T being determined by the parameter n. We seek the analyst's feedback on all the relations in T (line 2). Subsequently, we split T into T^+ and T^-, where T^+ contains the relevant relations and T^- contains the superfluous ones (line 3).

		IF1	IF2	IF3	IF4	IF5	IF6	IF7	IF8	DF1	DF2	DF3	DF4	DF5	DF6	DF7	DF8	DF9	DF10	DF11	DF12	DF13	DF14	DF15	DF16	Label
Current Training Data	**Rel 1.1**	A	A5	-1	0.5	1	2	F	F	0	1	2	0	1	0	0	1	2	0	0	0	0	0	1	2	R
	Rel 1.2	A	A3	-1	0.3	2	2	F	F	0	0	2	0	0	0	0	1	1	0	0	0	0	0	1	2	R
	Rel 1.3	G	G1	-1	0.15	1	2	T	F	0	0	1	0	0	0	1	0	0	0	0	0	0	0	2	1	S
	Rel 1.4	G	G1	-1	0.15	1	2	T	F	0	0	3	0	0	0	1	0	0	0	0	0	0	0	2	1	S
	Rel 2.1	A	A4	-1	0.27	1	2	F	F	0	1	2	0	1	0	0	1	2	0	0	0	0	0	0	0	R
	Rel 3.1	A	A1	0	0.21	1	1	T	F	0	1	2	0	0	0	0	0	1	0	0	1	0	1	1	1	S
	Rel 3.2	A	A6	1	0.5	1	3	F	F	0	1	2	1	0	0	0	0	1	1	0	1	0	0	1	1	S
	Rel 3.4	Ag	Ag1	-1	0.21	1	3	T	F	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	2	R
															+											
	Rel 3.3	A	A6	2	0.71	1	2	F	F	0	1	2	1	0	0	1	0	1	2	0	2	0	0	1	2	R

Fig. 8. Example feature matrix. Last column is the label (R for relevant and S for Superfluous). Highlighted cells increase by one when Rel 3.3 is added.

Once the relations in T have been labeled, we compute the feature functions $f_1, \ldots, f_l$ (line 4). As shown in Table VII, some of the features are *label-independent* and some are *label-dependent*. The former group is computed independently of user feedback (labels); whereas the latter requires knowledge of the labels in the training set. After computing the feature functions, we build the initial classifier $\mathcal{C}$ (line 5). To illustrate, we show in Fig. 8 the feature matrix (*FeatureMatrix* in the algorithm) that is built when the training data is composed of Rel 1.1 through Rel 1.4, Rel 2.1, Rel 3.1, Rel 3.2 and Rel 3.4 from the example relations of Fig. 1. Columns IF1–IF8 and DF1–DF16 are the features in Table VII. The last column is the label: Relevant (R) or Superfluous (S).

In the remainder of the algorithm (lines 6-24), we iteratively improve $\mathcal{C}$ as follows: we predict the labels for every relation in set X, i.e., $R \setminus T$, using $\mathcal{C}$ and without any feedback from the analyst (line 7). For each relation $r \in X$, the classifier predicts r as being superfluous with a probability of $p(r)$ (i.e, as relevant with a probability of $1 - p(r)$). We then compute two sets I^+ and I^-. These sets denote the automatic recommendations to be made to the analyst in the current iteration about relevance and superfluousness, respectively. The set I^+ is populated with relations that have been with high probability labeled as relevant in the current iteration (line 10). The set I^- has two parts to it: The first part is composed of the relations that have been repeatedly and with high probability labeled as superfluous (line 11). Recall the asymmetric treatment for predictions of relevance and superfluousness mentioned earlier in this section. The second part of I^- (line 12) is induced by LP groups (see Section 3.2.3).

Specifically, let L_{min} and L_{max} respectively denote the minimum and the maximum predicted $p(r)$ over all relations in X (lines 8 and 9). Further, let the parameter ϵ (resp. δ) be the margin for deciding whether, for a given relation r, the probability $p(r)$ is low enough (resp. high enough) with respect to L_{min} (resp. L_{max}) for the relation to make the cut as a prediction of "relevant" (resp. "superfluous"). We add a relation r to I^+ when $p(r) \leq L_{min} + \delta$ (line 10). We add r to I^- when $p(r) \geq L_{max} - \epsilon$, and further, r has been predicted with high probability to be superfluous in $k-1$ of the previous iterations (line 11). These $k-1$ iterations do not necessarily have to be consecutive. Additionally, we add to I^- any relation $r \in X$ such that r belongs to the same LP group as some $r' \in T^+$ (line 12). In other words, and in line with the findings in Section 3.2.3, once a relation from an LP group has been deemed relevant, i.e., the relation has landed in T^+, we recommend as superfluous all the remaining relations in that LP group which are yet to be inspected by the analyst.

Next, we present the recommendations in $I^+ \cup I^-$ to the analyst, if the union is non-empty. The analyst confirms and, when necessary, relabels those relations in I^+ and I^- where the automatic recommendations have been incorrect. Subsequently, T^+ and

T^- are extended with the revised I^- and I^+ (lines 16 and 17). This is of course after removing $I^+ \cup I^-$ from X (line 14).

Otherwise, if both I^+ and I^- are empty, we pick for presentation to the analyst and obtaining feedback a relation $r \in X$ for which we have the least amount of evidence in support of superfluousness (line 19). If there are multiple such relations, we pick one randomly. As noted in Section 2.2, picking element(s) over which the classifier is least certain and obtaining feedback over them is a common strategy (known as SAL).

Next, we add the analyst feedback on r to the training data (line 20), followed by removing r from X (line 21). We then rebuild $\mathcal{C}$ using the *updated* feature values for the extended training data (lines 22-23). This process continues until X becomes empty. We emphasize that on line 22, we update the values of label-dependent features for *all* those relations in $T^+ \cup T^-$ affected by the newly-added relations.

To illustrate, suppose a new relation, Rel 3.3 from Fig. 1, is labeled as relevant by the analyst and added to the training set of Fig. 8. Adding Rel 3.3 affects the cells highlighted green in the existing feature matrix. Specifically, the DF1 value for Rel 3.2 is increased by one as both Rel 3.3 and Rel 3.2 originate from requirement R3 (Fig. 1) and were extracted by the same rule (A6). Further, Rels 1.1, 2.1, 3.1 and 3.2 share an endpoint, “CMP”, with Rel 3.3. The DF2 and DF3 values for these relations thus need to be recomputed. The DF6 values for Rels 3.1 and 3.2 increase by one, since these relations share the same verb, “store”, with Rel 3.3. The DF13 values for Rels 3.1, 3.2 and 3.4 are updated as well, because the end concepts and the relation labels of these relations are all contained in Rel 3.3. DF15 is a generic version of DF1 and counts only the relations that originate from the same requirement. Therefore, when Rel 3.3 is added, the DF15 values for Rels 3.1, 3.2 and 3.4 are updated. As a side remark, we note that Rel 3.3 is in the same LP group as Rel 3.1 and Rel 3.2. Had Rel 3.1 and Rel 3.2 not been in the training data already, they would have further received an automated recommendation of “superfluous” and been escalated to the analyst for review.

The algorithm shown in Fig. 7 uses a single classifier, $\mathcal{C}$. An alternative is to use *ensemble learning*, where multiple classifiers are used together. Ensemble learning is known to often lead to better results than when classifiers are used individually [Miner et al. 2012]. The duality between our features for measuring relevance, i.e., DF1–DF6, DF13 and DF15 in Table VII, and those for measuring superfluousness, i.e., DF7–DF12, DF14 and DF16 in the table, leads to a natural two-classifier ensemble, made up of $\mathcal{C}_1$ and $\mathcal{C}_2$ defined as follows: $\mathcal{C}_1$ uses all the label-independent features, i.e., IF1–IF8, alongside the following subset of label-dependent features: DF1–DF6, DF13 and DF15. $\mathcal{C}_2$ uses all the label-independent features, just like in $\mathcal{C}_1$, but alongside the following label-dependent features: DF7–DF12, DF14 and DF16.

Classifiers $\mathcal{C}_1$ and $\mathcal{C}_2$ would still be applied to the relations in X, as shown in Fig. 7. This arrangement nevertheless yields, for each relation $r \in X$, two prediction probabilities, $p_1(r)$ and $p_2(r)$, indicating the probability of r being superfluous according to $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. In an ensemble setting, the computations on lines 10 and 11 need to be done by *consensus*. A relation is labeled as relevant (resp. superfluous) if both $\mathcal{C}_1$ and $\mathcal{C}_2$ predict it to be relevant (resp. superfluous). Otherwise, the prediction is inconclusive. To do so, we need to compute L_{min} and L_{max} (lines 8 and 9) separately for the two classifiers. Let $L_{1,min}$ and $L_{1,max}$ be these values for $\mathcal{C}_1$, and let $L_{2,min}$ and $L_{2,max}$ be these values for $\mathcal{C}_2$. In the ensemble setting, lines 10 and 11 of the algorithm of Fig. 7 need to be modified as follows:

$$I^+ = \{r \in X \mid p_1(r) \leq L_{1,min} + \delta \wedge p_2(r) \leq L_{2,min} + \delta\}$$
$$I^- = \{r \in X \mid (p_1(r) \geq L_{1,max} - \epsilon \wedge p_2(r) \geq L_{2,max} - \epsilon) \wedge$$
$$(r \text{ has been predicted with high probability to be superfluous in } k-1 \text{ of of the previous iterations})\}$$

The rest of the algorithm in Fig. 7 remains as is for ensemble learning. In Section 5, we provide a systematic empirical analysis of our algorithm, where we simultaneously consider the influence of different factors, including single versus ensemble learning as well as different choices for parameter values.

5. EMPIRICAL EVALUATION

In this section, we evaluate the filtering approach presented in Section 4. To do so, we use as case study material the survey results discussed in Section 3.

5.1. Research Questions (RQs)

Our evaluation aims to answer the following RQs:

RQ1. Which ML classification technique yields the most accurate recommender? Our recommender (algorithm of Fig. 7) can be realized using several alternative ML classification techniques. RQ1 identifies the most accurate alternative.

RQ2. Which ML features are the most influential for our recommender? In Table VII, we presented the complete set of features that we consider for classification. RQ2 assesses the importance of these features.

RQ3. What is the optimal tuning of our recommender? The algorithm of Fig. 7 has four parameters that need to be specified. In addition, one needs to select the feature functions to use for classification, and decide whether to apply ensemble classification (discussed in Section 4). RQ3 explores the impact of the parameter values, feature selection and ensemble learning on accuracy, and provides the best tuning of our recommender within the space of alternatives considered.

RQ4. Is our recommender accurate enough to be used in practice for identifying superfluous relations? As explained in Section 4, the main value of our recommender is in helping analyst find the superfluous relations. With RQ3 having established the optimal tuning of the recommender, RQ4 examines the overall accuracy of our automatic recommendations of superfluousness.

RQ5. Does our recommender have practical execution time? Our recommender has a human feedback loop in it. The main computational steps of the recommender (lines 7, 22, 23 of the algorithm of Fig. 7) therefore have to execute quickly enough to allow user interaction. RQ5 investigates the execution time of the recommender's main computational steps.

5.2. Implementation

We have implemented the algorithm of Fig. 7 using a combination of Java and R (https://www.r-project.org). More precisely, the features in Table VII are computed using Java code. The computed features are then passed to R for building ML-based classifiers. Data exchange between Java and R is handled via R's `Rserve` package (http://www.rforge.net/Rserve/). As per our evaluation outcomes, which we present momentarily, the best-performing ML classification technique in our context is Random Forest. The default ML technique in our implementation is R's `randomForest` package (https://www.stat.berkeley.edu/~breiman/RandomForests/). This choice is user-configurable;

other ML techniques can be used if desired. Our implementation is ≈1,500 lines of Java code and ≈200 lines of R scripts, excluding comments and third-party libraries. To facilitate replication, we make our implementation and non-proprietary case study material (Case B) publicly available at: https://sites.google.com/site/svvredomex/.

5.3. Experimental Setup

To answer the RQs in Section 5.1, we performed experiments EXPI, EXPII and EXPIII, described below.

EXPI. This experiment answers **RQ1**. We compare the accuracy of five alternative ML classification techniques. These are: *Random Forest*, *Logistic Regression*, *Decision Tree*, *Support Vector Machine* and *Artificial Neural Network* [Louridas and Ebert 2016]. To do so, we first subject these techniques to hyperparameter optimization using MultiSearch [Weka MultiSearch 2017]. The optimization is performed over the prediction accuracy metric (see Section 5.4) and across our case studies, Case A, Case B and Case C, in Table II. EXPI is based on the complete set of features presented in Table VII.

Following hyperparameter optimization, we perform a standard 10-fold cross validation [Miner et al. 2012] of the five classification techniques considered. Specifically, the set of expert-reviewed relations in each case study is randomly divided into ten equal subsets (folds). We select one subset as the test set, and the remaining nine subsets as the training set. We compute the feature matrix for the training set (as explained in Section 4), build a classifier based on the training set, and apply the classifier to the test set. We repeat this process ten times to obtain the prediction results of each of the ten classifiers on their respective test sets.

EXPII. This experiment answers **RQ2**. For most ML classification problems, the process through which the features are engineered is informal and heavily reliant on the domain knowledge and the intuitions of the individuals who define the features [Kasun et al. 2013]. Consequently, one often has no definitive way of knowing in advance whether a feature is going to be instrumental for building a good prediction model. The importance of the features is usually established after the fact using statistical measures. In EXPII, we employ the Mean Decrease in Accuracy (MDA) metric, discussed in Section 5.4, for ranking the features of Table VII based on their importance. The resulting ranking is further used in RQ3 for comparing the accuracy of our recommender using all features versus important features only.

EXPIII. This experiment answers **RQ3–RQ5** by applying the algorithm of Fig. 7 with the following configuration options:

(1) *Features*: We consider two options for the set of features to use for building $\mathcal{C}$. The first option is to use all the features in Table VII; the second option is to use only those features that are found to be important in EXPII. We refer to the first option as **all** and to the second as **reduced**. What features **reduced** is exactly composed of is discussed after presenting the results of EXPII and answering RQ2.
(2) *Learner*: We consider two alternative ways of building $\mathcal{C}$: **single** learner and **ensemble** learner (see Section 4).
(3) *Size of initial training set (parameter n in Fig. 7)*: For the size of the initial training set T, we consider five levels: **10%**, **20%**, **30%**, **40%** and **50%** of $|R|$, where R is the set of all relations in a given case study. We recall from Table II that $|R| = 213$ in Case A, $|R| = 157$ in Case B, and $|R| = 101$ in Case C. We do not set n to a value lower than 10% because we need a reasonable amount of training data to build a classifier that can make meaningful predictions. At the same time, we do not want to set n to a higher level than 50%, as such a value would both imply a large upfront

effort from the analyst for labeling the training set, and further leave a small pool of relations for automated predictions. For comparing our three case studies, which have different numbers of relations, we use the *absolute* values of n at different proportional levels.

(4) *Probability margin for predicting relevant relations (parameter δ in Fig. 7)*: We consider two levels for δ: **5%** and **10%**. We do not consider δ values that are too small or too large. A too large value of δ may rule out correct recommendations of superfluousness; a too small value may lead to more relevant relations getting wrongly predicted as superfluous.

(5) *Probability margin for predicting superfluous relations (parameter ϵ in Fig. 7)*: We consider two levels for ϵ: **5%** and **10%**. We do not consider ϵ values that are too small or too large. A too small value of ϵ may lead to fewer superfluous recommendations, and a too large value of ϵ may lead to more incorrect recommendations of superfluousness.

(6) *Number of times a relation needs to be predicted as superfluous before it is recommended to the user (parameter k in Fig. 7)*: We consider five levels for k. These are: an absolute value of **one**, alongside four proportional levels, **5%**, **10%**, **15%** and **20%**. The proportional levels are relative to the total number of relations in X, as defined on line 1 of the algorithm in Fig. 7. When the level denoted **one** is used, a prediction of "superfluous" over a relation immediately prompts a recommendation of "superfluous". The remaining levels for k signify different amounts of evidence that must build up in support of superfluousness, before a recommendation of "superfluous" is made. For example, if $|X| = 200$, the **5%** level would require that a relation should be predicted as superfluous for 10 times, before the relation is recommended as being so.

With the different levels defined for the above configuration options and parameters, we obtain $2 \times 2 \times 5 \times 2 \times 2 \times 5 = 400$ different configurations of the algorithm in Fig. 7 over each of our three case studies. To account for the randomness of our algorithm, in particular the selection of the initial training set, we run each configuration ten times. This yields $400 \times 10 = 4000$ runs per case study, i.e., $4000 \times 3 = 12000$ runs in total.

Running EXPIII requires feedback from the analyst on lines 2, 15 and 19 of the algorithm of Fig. 7. This feedback is simulated based on the survey results (gold standards) presented in Section 3.

5.4. Metrics

For EXPI, we evaluate the alternative classification techniques under analysis via the *prediction accuracy* metric. Prediction accuracy is computed as the ratio of correct predictions over the total number of predictions.

For EXPII, we measure feature importance using Mean Decrease in Accuracy (MDA) [Breiman et al. 1984]. We selected this metric in light of the results of RQ1. In particular, as we argue in Section 5.5, Random Forest turns out to be the overall best classification technique for our recommender. MDA is one of the main methods used for ranking the importance of features in Random-Forest-based classification models [Strobl et al. 2009]. Intuitively, MDA represents the loss of predictive power resulting from the exclusion of a given feature from the classification model. Naturally, the higher the MDA score for a given feature, the more important the feature is. The standard practice with Random Forest is to discard features that have a negative, zero, or very low positive MDA score [Strobl et al. 2009].

For EXPIII, we compute the precision and recall of the recommendations of superfluousness. We denote (1) by A the number of correct recommendations of superfluousness made on line 15 of the RECOMMENDER algorithm in Fig. 7, (2) by B the number of all

Table VIII. Accuracy results for EXPI.

Case	Random Forest	Logistic Regression	Decision Tree	Support Vector Machine	Artificial Neural Network
Case A	89.2%	69.0%	86.4%	80.3%	71.8%
Case B	89.1%	69.4%	92.4%	75.2%	64.3%
Case C	92.1%	56.4%	82.2%	77.2%	60.4%

recommendations of superfluousness made on line 15, be the recommendations correct or incorrect, (3) by C the number of superfluous relations in R on line 1, i.e., the set of all superfluous relations in a given case study, and (4) by D the number of superfluous relations in set X, i.e., the set of all relations minus the initial training data, i.e., T on line 1.

We compute *Precision* as A/B. For recall, we consider two different metrics. The first metric, denoted simply *Recall*, is computed as A/C. *Recall* does *not* exclude the set of relations T used for initial training. Any superfluous relation that happens to be in T is thus counted as a miss by the *Recall* metric. We define a second recall metric, denoted *Recall*$_X$ and computed as A/D. *Recall*$_X$, in contrast to *Recall*, excludes T, thus focusing only on relations that have a chance of being assigned an automatic recommendation by the RECOMMENDER algorithm. Our answer to RQ3 is based on *Precision* and *Recall*, whereas our answer to RQ4 is based on *Precision* and *Recall*$_X$.

5.5. Discussion

Below, we present our answers to the RQs of Section 5.1.

RQ1. We answer RQ1 by performing EXPI and computing prediction accuracy for the five ML classification techniques under investigation. Table VIII shows for each case study and each ML technique the mean accuracy of 10-fold cross validation, obtained with optimized hyperparameters for the underlying ML technique. As highlighted in the table, for Case A and Case C, Random Forest outperforms all the other techniques. In Case B, Decision Tree turns out to be the most accurate, but is closely followed by Random Forest (accuracy difference of 3.3%).

The answer to **RQ1** is that *Random Forest* performs consistently well across our three case studies. We thus use Random Forest for answering the remaining RQs.

RQ2. To answer RQ2, we conduct EXPII. Fig. 9 shows, for each case study, the features ranked in descending order of importance as computed by the MDA metric introduced in Section 5.4. Based on the rankings, DF9, DF3, IF2 and IF4 are consistently among the top five most important features. While it is difficult to provide a precise explanation as to why some features are more influential than others, we present our intuition about the obtained results. The high importance of DF3 and DF9 is likely an indication that the relevance of a given relation strongly correlates with the experts' perception about other relations that share an endpoint with the relation in question.

With regard to IF2, we observe from Table IV that although all the model extraction rules are overall useful, there are certain rules that have poor relevance in certain case studies. For instance, rules Ag2 and At1 in Case A, and rule G1 in Case B and Case C yield no or few relevant relations. ML classification techniques are quite adept at picking up on such patterns. This likely contributes to the importance of IF2.

Finally, IF4 is related to the number of tokens in a relation relative to the length of the underlying requirement. The importance of this feature is likely due to the fact that relations containing a significant amount of textual content have a high chance of

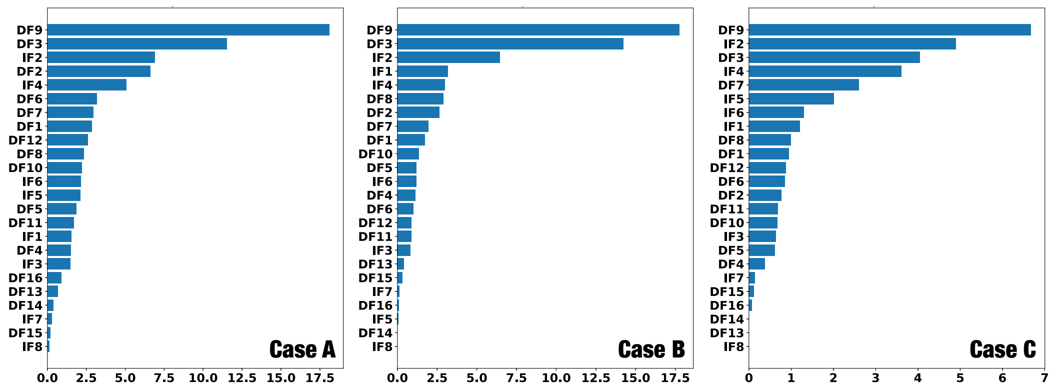


Fig. 9. Features of Table VII ranked in descending order of Mean Decrease in Accuracy (MDA).

being superfluous. This is natural: such relations defeat the purpose of domain model construction; one may as well read the requirements than consult the domain model to obtain the information content.

With the influence of features determined, a standard practice in ML is to discard features with a negative or only marginally positive impact on prediction accuracy. This is often referred to as feature reduction (or feature selection). The main benefits of feature reduction are: mitigating overfitting risks, simplifying the resulting prediction models, improving training times, and avoiding the undesirable consequences of using too many features (the curse of dimensionality) [Guyon and Elisseeff 2003].

In our work, we consider all features with an MDA score of $< 1\%$ across all case studies as candidates for removal from the feature set. The features that fall below the $< 1\%$ in all Case A, Case B, and Case C are the following: DF13, DF14, DF15, DF16, IF7 and IF8. For EXPIII, we define the **reduced** option to be the set of all features with the above-mentioned six features removed.

The answer to **RQ2** is that, among the features in Table VII, the most important ones are DF3, DF9, IF2 and IF4, and the least important ones are DF13, DF14, DF15, DF16, IF7 and IF8.

RQ3. To answer this question, we perform EXPIII and calculate the *Precision* and *Recall* metrics, defined in Section 5.4, for 12000 full executions (4000 from each case study) of the RECOMMENDER algorithm in Fig. 7. Our analysis in RQ3 is based on the mean values for the ten runs of each configuration of the algorithm. This gives us a total of $400 \times 3 = 1200$ datapoints.

We are interested in configurations that yield high precision, i.e., recommend few relevant relations as superfluous. Poor precision means that the recommendations are not trustworthy. If the analyst trusts the recommendations when precision is low, she will risk filtering useful information. On the other hand, if she attempts to vet all the recommendations in detail, any potential effort savings brought about by the algorithm will be canceled. To identify configurations with high precision across the three case studies, we use regression trees [Breiman et al. 1984]. A regression tree is constructed by partitioning a data set in a step-wise manner in order to obtain partitions that minimize variation with respect to a criterion, e.g., precision or recall.

The first column of Fig. 10 shows the regression trees for precision in our three case studies. At each node, the tree identifies the parameter that is most influential in explaining the variation in precision, and partitions the node based on that parameter. We show the number of elements, mean and standard deviation of precision at each node, as well as the difference between the mean precision scores of the children of

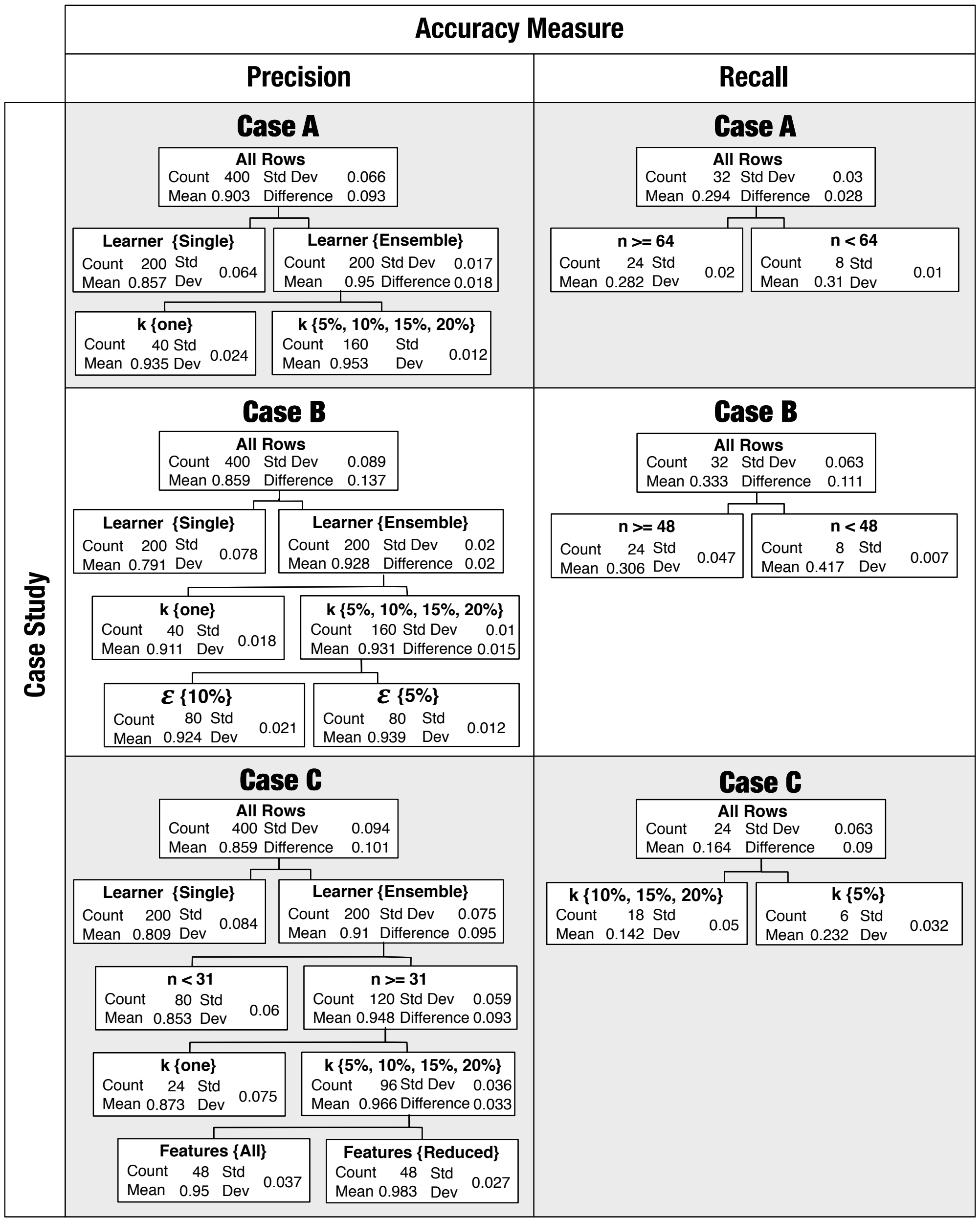

Accuracy Measure
Precision
Recall
Case Study
Case A
All Rows Count 400 Std Dev 0.066 Mean 0.903 Difference 0.093
Learner {Single} Count 200 Mean 0.857 Std Dev 0.064
Learner {Ensemble} Count 200 Std Dev 0.017 Mean 0.95 Difference 0.018
k {one} Count 40 Mean 0.935 Std Dev 0.024
k {5%, 10%, 15%, 20%} Count 160 Mean 0.953 Std Dev 0.012
Case A
All Rows Count 32 Std Dev 0.03 Mean 0.294 Difference 0.028
n >= 64 Count 24 Mean 0.282 Std Dev 0.02
n < 64 Count 8 Mean 0.31 Std Dev 0.01
Case B
All Rows Count 400 Std Dev 0.089 Mean 0.859 Difference 0.137
Learner {Single} Count 200 Mean 0.791 Std Dev 0.078
Learner {Ensemble} Count 200 Std Dev 0.02 Mean 0.928 Difference 0.02
k {one} Count 40 Mean 0.911 Std Dev 0.018
k {5%, 10%, 15%, 20%} Count 160 Std Dev 0.01 Mean 0.931 Difference 0.015
ℰ {10%} Count 80 Mean 0.924 Std Dev 0.021
ℰ {5%} Count 80 Mean 0.939 Std Dev 0.012
Case B
All Rows Count 32 Std Dev 0.063 Mean 0.333 Difference 0.111
n >= 48 Count 24 Mean 0.306 Std Dev 0.047
n < 48 Count 8 Mean 0.417 Std Dev 0.007
Case C
All Rows Count 400 Std Dev 0.094 Mean 0.859 Difference 0.101
Learner {Single} Count 200 Mean 0.809 Std Dev 0.084
Learner {Ensemble} Count 200 Std Dev 0.075 Mean 0.91 Difference 0.095
n < 31 Count 80 Mean 0.853 Std Dev 0.06
n >= 31 Count 120 Std Dev 0.059 Mean 0.948 Difference 0.093
k {one} Count 24 Mean 0.873 Std Dev 0.075
k {5%, 10%, 15%, 20%} Count 96 Std Dev 0.036 Mean 0.966 Difference 0.033
Features {All} Count 48 Mean 0.95 Std Dev 0.037
Features {Reduced} Count 48 Mean 0.983 Std Dev 0.027
Case C
All Rows Count 24 Std Dev 0.063 Mean 0.164 Difference 0.09
k {10%, 15%, 20%} Count 18 Mean 0.142 Std Dev 0.05
k {5%} Count 6 Mean 0.232 Std Dev 0.032

Fig. 10. Regression tree for *Precision* and *Recall*.

non-leaf nodes. A node is expanded if the difference between the means of its children nodes is >0.01 (i.e., $>1\%$), which we deem to be the minimum difference of practical relevance.

As indicated by the precision column in Fig. 10, in all three case studies, the most critical decision is the choice of the learner. The ensemble learner performs consistently better than the single learner. The second most critical decision in Case A and Case B (and the third most critical decision in Case C) is the choice of parameter k.

The 5%, 10%, 15% and 20% levels perform significantly better than the "one" level. This finding indicates that a single prediction of superfluousness is insufficient for escalating the prediction into a recommendation of superfluousness.

In addition to the two parameters discussed above, which appear in the precision regression trees of all the three case studies, there are three other parameters, one of which is influential in Case B and two of which are influential in Case C. In Case B, the 5% level for ϵ is significantly better than the 10% level. In Case C, we need to seed the initial training set with at least 31 relations. Furthermore, using the reduced set of feature functions in Case C provides an edge, compared to when all the feature functions are used.

Ultimately, our goal is to pick configurations that perform consistently well across all three case studies. That is, we are interested in the intersection of the best configurations. Since no inconsistency exists between the precision regression trees[1], the intersection of the configurations that perform best with respect to precision is non-empty. Specifically, the configurations falling in the intersection are the following:

(1) **Features:** {Reduced};
(2) **Learner:** {Ensemble};
(3) n**:** ≥ 31, {20%, 30%, 40%, 50%} in Cases A and B, and {30%, 40%, 50%} in Case C;
(4) δ**:** {5%, 10%};
(5) ϵ**:** {5%};
(6) k**:** {5%, 10%, 15%, 20%}.

All the above configurations, i.e., 32 ($4 \times 2 \times 4$) configurations in Case A, 32 in Case B, and 24 ($3 \times 2 \times 4$) in Case C – lead to a mean precision of $\geq 95\%$. These configurations are thus all trustworthy for making recommendations of superfluousness.

Having identified the configurations that yield the highest precision, we now shift our attention to recall. The recall regression trees are shown in the second column of Fig. 10. In contrast to the precision regression trees (first column of Fig. 10), which cover all configurations, the recall regression trees are restricted to the configurations that yield the best precision. We remember, from Section 5.4, the two different notions of recall defined. The recall regression trees in Fig. 10 are based on the *Recall* metric. While one would conventionally use $Recall_X$ (as we will do in RQ4) for measuring the recall of a trained classifier, this metric is not suitable for tuning the parameters of our recommender. The reason is that $Recall_X$ would not account for the training that goes into building an accurate classifier. Regression tree analysis, if performed over $Recall_X$, would thus unduly favor larger training sets. In contrast, *Recall* allows us to maximize the net gain from the recommender, taking into account the amount of training required.

As indicated by the recall regression trees, the most critical decision for recall is the choice of n in Case A and Case B. This result simply highlights the fact that if the initial training set is too large, there will not be enough iterations of the algorithm to monitor the predictions and issue recommendations. In Case C, the most critical decision with respect to recall is the choice of k. In particular, $k = 5\%$ leads to better recall than larger k values. This indicates that if some relation r is predicted as superfluous for $k = 5\% \times |X|$ times (where X is as defined on line 1 of the algorithm in Fig. 7), then the evidence in support of r being superfluous is strong enough to warrant making a recommendation to the analyst. Higher k values, although strengthening the confidence, would negatively impact recall, since they allow fewer predictions to mature to recommendations.

[1]An example inconsistency would have been if, say, Case A favored Ensemble learning while Case B favored Single learning.

Table IX. Optimal configuration and associated accuracy results.

Case	Features	Learner	n	δ	ε	k	Precision	Recall	$Recall_X$
Case A	Reduced	Ensemble	43 (20%)	5%	5%	5%	95.2%	35.0%	43.6%
Case B	Reduced	Ensemble	32 (20%)	5%	5%	5%	95.1%	43.0%	50.1%
Case C	Reduced	Ensemble	31 (30%)	5%	5%	5%	98.0%	27.3%	40.0%

We note that similar to the precision regression trees, no inconsistencies were seen in the recall regression trees. We further observe the following from the precision and recall regression trees, when they are taken together: the choice of δ (**5%** versus **10%**) has no significant impact. A further examination indicated that our algorithm has little sensitivity to this parameter, unless it is set to a really low ($< 1\%$) or a really high ($> 30\%$) value. Neither action was justified, given the intuition behind δ, stated earlier. Between the two levels considered in our experiment, we pick **5%**, thus favoring recall over precision, noting that precision is already very high.

The answer to **RQ3**, i.e., the optimal configuration, is shown in Table IX. For n, we show the smallest level in each case study with ≥ 31 relations. For Case A and Case B, this is the 20% level, and for Case C, the 30% level. Table IX further shows for each case study the optimal precision and recall (average of the ten runs of the optimal configuration). The practical utility of these results is discussed in **RQ4**.

RQ4. From Table IX, we observe that using the *Recall* metric, we obtain a recall range of 27% to 43%, with precision being consistently above 95%. If we exclude the training data from our analysis, i.e., apply the $Recall_X$ metric, the recall range is between 40% to 50%. High precision provides confidence that the recommendations are very likely to be correct. Since analysts want to dispose of superfluous relations with as little effort as possible, such trustworthy recommendations appear useful. The amount of effort that analysts save will, of course, depend on recall. The question that remains is which of the two recall metrics is more representative of the effort savings in practice. As we discussed in RQ3, the initial training size in the best configurations is in the range of 30 to 40 elements. Had we analyzed the underlying requirements documents in full, the size of the initial training set would have constituted a smaller fraction of the entire set of relations set size. We therefore believe that $Recall_X$ is more indicative of the effort savings brought about by our approach.

Indeed, we anticipate even larger savings if the requirements documents are analyzed in their entirety. In particular, we examined how much of the vocabulary of the full requirements documents was covered by the document segments in our case studies. To this end, we measured the number of distinct noun phrases (NPs) and verb phrases (VPs) in the segments as a ratio of the number of NPs and VPs in the full documents. We observed that, while the segments accounted for an average of $\approx 32\%$ of the total number of requirements, the NPs and VPs in these segments covered on average $\approx 61\%$ of the full set of NPs and VPs. This observation suggests that we are likely to see a saturation in vocabulary. With larger document segments analyzed, such saturation would increase the quality of the predictions and consequently the proportion of automated recommendations.

The answer to **RQ4** is that, once trained, our recommender automatically detects an average of $\approx 45\%$ of the superfluous relations with a precision of $\approx 96\%$. Since precision is very high, the automatic recommendations are trustworthy. In other words, analysts can dispose of nearly half of the superfluous relations with minimal manual effort. Our results are particularly promising when one considers the subjectivity phenomenon described in Section 3.2.4, and the fact that such subjectivity poses limits on how far an automated approach can go in detecting superfluousness.

RQ5. The most important consideration with regard to the execution time is the following: once the analyst has provided some feedback, how long does it take the algorithm to update the feature matrix, rebuild the classifier and compute fresh predictions? For our largest case study (Case A), in the worst case, this cycle took $\approx$2 seconds on a laptop with a 2.3 GHz CPU and 8GB of memory. Our case studies however cover only about one third of the underlying requirements documents (see Table II). Furthermore, for many systems, the requirements documents may be larger than those in our case studies.

To gain a better understanding of execution time, we artificially duplicated the relations in Case A ten times, obtaining a total of 2130 relations. For this increased data set, the worst execution time for one iteration of our algorithm increased to $\approx$6 seconds. When handling large sets of relations, a simple strategy for maintaining the interactiveness of our algorithm would be to ensure that, in each iteration of the algorithm, we present to the analyst at least a minimum number of relations, e.g., 1% of the total number of relations. More precisely, if the number of relations in $I^+ \cup I^-$ falls below the minimum threshold in a given iteration of the algorithm, we can proceed to additionally request user feedback on as many uncertain relations as necessary to meet the threshold. By doing so, we can both reduce the number of iterations of the algorithm, and further allow the computational tasks to be parallelized with user interactions, thus minimizing interaction delays.

The answer to **RQ5** is as follows: for small sets of relations (100-500 relations), the algorithm of Fig. 7 can be run as-is. For larger sets, one can ensure that the number of relations subject to feedback on lines 15 and 19 of the algorithm of Fig. 7 is above a predefined minimum threshold. Further, some of the computational tasks can be run in the background while the user is reviewing the relations.

6. THREATS TO VALIDITY

Internal Validity. The gold standards built for relevance in Section 3 are based on experts examining the requirements and the extracted relations one at a time. We did not present to the experts the extracted domain models in their entirety. We thus cannot completely rule out the possibility that the experts may have made different decisions about relevance, had they seen the global picture, i.e., all the extracted relations at once. We decided against showing the extracted domain models in their entirety in order to avoid confounding factors due to potentially poor model layout and cognitive overload. We believe the way we conducted our surveys does not pose a substantial internal validity threat, since the experts could easily make up their mind about relevance based on the information content they saw in the individual requirements.

Construct Validity. Our metrics do not account for the tacit (implicit) knowledge that is required for building a domain model, but which cannot be inferred from the requirements. This does not pose a threat to construct validity in our context, since our focus was on filtering useless information that is *explicit* in the requirements and extractable via NLP.

Conclusion Validity. The gold standard for each of our case studies – Case A, Case B and Case C – is based on feedback from an individual respondent, noting that any potential respondent had to be a domain expert. In each Case A and Case B, we were unable to recruit more than one domain expert. In Case C, we collected feedback from three experts for a part of the case study, but as stated earlier, only one expert participated throughout the entire case study. To mitigate threats to conclusion validity, we considered three distinct systems with three distinct experts. Further, each expert covered many (> 100) relations to minimize potential expert errors.

External Validity. Our evaluation involved three industrial case studies from different domains. Our conclusions are based on the combined results of these case studies. The consistency seen across the results provides confidence about the generalizability of our approach. Future case studies of larger sizes are nevertheless necessary for improving external validity.

7. RELATED WORK

In this section, we position our work within the state of the art, and compare with research strands that relate most closely to our approach. We organize our discussion under three headings: (1) model extraction, (2) superfluousness in NLP results, and (3) applications of ML in Requirements Engineering.

7.1. Model Extraction

This article was not meant at developing a new approach for domain model extraction. We thus do not compare the model extractor we build upon in this article against other existing solutions. For such a comparison, consult [Arora et al. 2016]. What is pertinent to this article from the literature on domain model extraction are the existing empirical results. To our knowledge, relevance – the main focus of our analysis in Section 3 – has not been considered in previous work on model extraction. Indeed, we are not aware of any prior strands of work where model extraction results have been validated directly with domain experts. This potentially explains why relevance has been overlooked.

The results of Section 3 further show that Link Paths (LP) are indeed useful for building domain models. We are not aware of any model extractor except ours that accounts for LP relations. The topic of indirect relations, including LP, is the subject of research in the NLP community. Recent versions of the Stanford CoreNLP toolkit [Manning et al. 2014] provide a module, named OpenIE (Open domain Information Extraction) [Angeli et al. 2015], for extracting both direct and indirect relations. The development of this module signifies the broader usefulness of indirect relations.

7.2. Superfluousness in NLP Results

Superfluousness is a recurring issue when NLP is used for information extraction [Manning and Schütze 1999; Jacquemin 2001]. This issue has been attributed to two causes: (1) NLP's inability to differentiate the specific information required by the users from other information in a given text [Krauthammer and Nenadic 2004; Jackson and Moulinier 2007; Nikfarjam et al. 2015], and (2) the subjectivity involved in determining what extracted information is relevant [Kosala and Blockeel 2000; Scholz and Conrad 2013]. Our results in Section 3 (specifically, the reasons for superfluousness shown in Table V and the discussion about interrator agreement in Section 3.2.4) provide evidence that both causes are pertinent to NLP-based model extraction.

In the field of Requirements Engineering, the manual effort associated with filtering the superfluous information produced by NLP has often been considered a fair price to pay in exchange for the ability to extract (nearly) all the relevant information [Mahmoud and Williams 2016]. The central premise for our work in this article is that analysts can benefit from automated assistance in filtering the superfluous information, thus making NLP an even more compelling choice for requirements analysis.

There are some recent threads of work in which the problem of filtering superfluous information is explicitly tackled. Rago et al. [2016] propose a query language to help analysts identify false positives in cross-cutting concerns that have been extracted from requirements using NLP. Bhatia et al. [2016] use crowd-workers for filtering superfluous privacy goals extracted from privacy policies. Quirchmayr et al. [2017] develop a filter based on predefined phrase patterns for finding superfluous software features extracted from user manuals.

The above approaches are not a suitable match for our application context. First, due to the extensive level of experience required for domain model construction, crowd-workers are unlikely to be able to contribute to this task in an effective manner. Further, confidentiality issues would complicate sharing proprietary system requirements with crowd-workers. Similarly, devising a complete set of generalizable criteria for distinguishing superfluous and relevant domain model elements is difficult, if not infeasible. Queries and heuristics, e.g., as employed by Rago et al. [2016] and Quirchmayr et al. [2017], can be helpful for specific domains and document types; however, these approaches cannot adapt themselves to the reasoning applied by experts in a domain that has not been studied a priori. In contrast, our approach, which builds on ML, can mimic the logic applied by experts in any domain, without the need for this logic to be made explicit and articulated.

7.3. Applications of ML in Requirements Engineering

ML has been considered for automating a variety of Requirements Engineering tasks. The tasks to which ML has been applied the most are requirements identification and classification. Hayes et al. [2014] propose an ML toolkit for requirements assessment, and demonstrate the application of the toolkit for requirements classification along different dimensions, e.g., functional versus non-functional and temporal versus non-temporal. Cleland-Huang et al. [2007] develop an iterative classifier, based on information retrieval techniques, for automated identification and classification of non-functional requirements. Kurtanović and Maalej [2017] propose an approach for distinguishing functional and non-functional requirements using Support Vector Machines. Casamayor et al. [2010] use supervised and unsupervised (clustering) techniques for identifying non-functional requirements in textual specifications. Rodeghero et al. [2017] compare Logistic Regression and Support Vector Machines for identifying requirements-related information in the transcripts of developer-client conversations. Maalej et al. [2016] and Kurtanović and Maalej [2018] explore several ML techniques, e.g., Naive Bayes [Witten et al. 2016], for extracting feature requests and user rationale from reviews. Winkler and Vogelsang [2016, 2018] use deep learning techniques for classifying requirements and auxiliary content in textual descriptions.

Requirements traceability detection is another task where ML is gaining increasing traction. Cleland-Huang et al. [2010] adopt the classifier developed in their earlier work [Cleland-Huang et al. 2007] for generating trace links from regulatory codes to requirements. Guo et al. [2017] use deep learning techniques for trace generation from requirements to downstream development artifacts. Wang et al. [2018] propose an approach for enhancing the accuracy of automated requirements traceability using Artificial Neural Networks.

Using ML has been further studied, albeit to a more limited extent, for other requirements analysis activities. For example, Yang et al. [2010] apply Logistic Regression for detecting requirements ambiguities; and Perini et al. [2013] employ boosting techniques [Witten et al. 2016] for requirements prioritization.

None of the research strands outlined above use ML for addressing the same problem as what we tackle in this article, namely assisting analysts with filtering superfluous information during domain modeling.

8. CONCLUSION

We proposed and evaluated an active learning approach for filtering superfluous elements from the output of domain model extraction tools. Our empirical results over industrial case studies indicate that, on average, our approach filters 45% of superfluous domain model elements with a precision of 96%.

The features that we defined and employed for machine learning are targeted specifically at detecting the relevance and superfluousness of domain model elements. We found such specificity to be important, as devising an effective solution necessitated that we carefully take the problem context into account, and exploit as much as possible the assumptions and intuitions valid in this context. At the same time, the general idea behind our work, namely applying active learning to provide automated decision support, can be useful for other requirements analysis tasks, e.g., glossary construction, traceability link vetting, and inconsistency handling. To this end, our technical solution provides a concrete instantiation of the active learning process, covering feature specification and selection, parameter tuning, user feedback simulation, and accuracy analysis. We thus believe that our work in this article can offer value beyond domain model construction by paving the way for developing other human-in-the-loop requirements automation techniques.

For future work, we would like to further improve our approach in terms of the proportion of superfluous elements it can identify, while maintaining the high precision already achieved. To this end, we are investigating ways for transfer learning [Pan and Yang 2010], which enable the reuse of labeled data acquired from one project for making predictions in other (unlabeled) projects. Conventional transfer learning techniques are not readily applicable in our context, since, first, we cannot compute our label-dependent features for a given set of relations without the analyst having provided feedback on that particular set, and second, the label-dependent features rely indirectly on the requirements terminology. While these characteristics may render cross-domain knowledge reuse infeasible, we still need to investigate whether transfer learning would be feasible for requirements documents within the same domain.

Another area for potential improvement has to do with how we pick the relations to include in the initial training set. Our current recommendation algorithm uses a random sampling strategy for this purpose. This decision was made after we tried, without success, a number of other sampling strategies, where the initial training set was picked according to some optimization criterion, e.g., covering as many frequently recurring concepts as possible, or maximizing the diversity of the terminology used within the selected relations. Further investigation is required to determine whether random sampling can be outperformed by a more systematic strategy.

To tune the hyperparameters of the ML classification techniques in our evaluation, we relied on a common but rather primitive strategy (MultiSearch). Noting the increasing evidence that hyperparameters can considerably affect the accuracy of ML in different contexts [Bergstra and Bengio 2012], we need to look more closely into alternative strategies for optimizing the hyperparameters, and determine whether the accuracy of our approach can be further increased using better hyperparameter values. Two interesting optimization strategies that we would like to examine in the future for this purpose are differential evolution [Fu et al. 2016] and sample-and-prune [Chen et al. 2018].

Finally, in our evaluation, we followed a binary logic for recommendations (i.e., recommended / not recommended). In reality, the analysts would also be interested in knowing, e.g., through color coding, how much evidence there is in support of superfluousness (or relevance), with an understanding that the evidence may be insufficient for reliable recommendations. A more conclusive evaluation of our approach which considers the above angles would require larger case studies and a human-in-the-loop realization of our recommendation algorithm.

Acknowledgment. This project has received funding from the Luxembourg National Research Fund (FNR) under grants FNR/P10/03 and FNR-11601446, and from the

European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 694277).

REFERENCES

Alan Akbik and Juergen Broß. 2009. Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns.. In *Workshop on Semantic Search at International World Wide Web Conference (WWW'09)*. ACM, Madrid, Spain, 5–16.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL'15)*. ACL, Beijing, China, 344–354.

Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2016. Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation. In *19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*. ACM, St. Malo, France, 250–260.

Carl Auerbach and Louise B Silverstein. 2003. *Qualitative data: An introduction to coding and analysis*. NYU press, New York, USA.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

Jaspreet Bhatia, Travis D. Breaux, and Florian Schaub. 2016. Mining Privacy Goals from Privacy Policies Using Hybridized Task Recomposition. *ACM Transactions on Software Engineering and Methodology (ACM TOSEM)* 25, 3 (2016), 22:1–22:24.

Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. 1984. *Classification and Regression Trees*. Wadsworth and Brooks, London, UK.

Agustin Casamayor, Daniela Godoy, and Marcelo Campo. 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology* 52, 4 (2010), 436–445.

Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2018. "Sampling" as a Baseline Optimizer for Search-based Software Engineering. *IEEE Transactions on Software Engineering* to appear, 00 (2018), 00. DOI:http://dx.doi.org/10.1109/TSE.2018.2790925

Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. 2010. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*. ACM, Cape Town, South Africa, 155–164.

Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated classification of non-functional requirements. *Requirements Engineering* 12, 2 (2007), 103–120.

Gordon V Cormack and Maura R Grossman. 2014. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, Gold Coast, Australia, 153–162.

Dave DeBarr and Harry Wechsler. 2009. Spam detection using clustering, random forests, and active learning. In *Sixth Conference on Email and Anti-Spam (CEAS)*. Citeseer, Mountain View, USA, 1–6.

Mosa Elbendak, Paul Vickers, and Nick Rossiter. 2011. Parsed use case descriptions as a basis for object-oriented class model generation. *Journal of Systems and Software (JSS)* 84, 7 (2011), 1209 – 1223.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Conference on Empirical Methods in Natural Lan-*

guage Processing (EMNLP'11). ACL, Edinburgh, UK, 1535–1545.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76, 5 (1971), 378–382.

Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.

Gonzalo Génova, José M Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. 2013. A framework to measure and improve the quality of textual requirements. *Requirements engineering* 18, 1 (2013), 25–41.

Jin Guo, Jinhui Cheng, and Jane Cleland-Huang. 2017. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In *IEEE/ACM 39th International Conference on Software Engineering (ICSE'17)*. IEEE, Buenos Aires, Argentina, 255–272.

Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, 1 (2003), 1157–1182.

Jane Huffman Hayes, Wenbin Li, and Mona Rahimi. 2014. Weka meets TraceLab: Toward convenient classification: Machine learning for requirements engineering problems: A position paper. In *1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'14)*. IEEE, Karlskrona, Sweden, 9–12.

Peter Jackson and Isabelle Moulinier. 2007. *Natural language processing for online applications: Text retrieval, extraction and categorization*. Vol. 5. John Benjamins Publishing, Amsterdam, Netherlands.

Christian Jacquemin. 2001. *Spotting and discovering terms through natural language processing*. MIT press, Cambridge, USA.

Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, Guang-Bin Huang, and Chi Man Vong. 2013. Representational learning with extreme learning machine for big data. *IEEE intelligent systems* 28, 6 (2013), 31–34.

Raymond Kosala and Hendrik Blockeel. 2000. Web mining research: A survey. *ACM SIGKDD Explorations Newsletter* 2, 1 (2000), 1–15.

Michael Krauthammer and Goran Nenadic. 2004. Term identification in the biomedical literature. *Journal of Biomedical Informatics (JBI)* 37, 6 (2004), 512–526.

Zijad Kurtanović and Walid Maalej. 2017. Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning. In *25th IEEE International Requirements Engineering Conference (RE'2017)*. IEEE, Lisbon,Portugal, 490–495.

Zijad Kurtanović and Walid Maalej. 2018. On user rationale in software engineering. *Requirements Engineering* 23, 3 (2018), 357–379.

J Richard Landis and Gary G Koch. 1977. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics* 33, 2 (1977), 363–374.

Craig Larman. 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall, New Jersey, USA.

David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th ACM International conference on Research and development in information retrieval (SIGIR'94)*. Springer, Dublin, Ireland, 3–12.

Odd Ivar Lindland, Guttorm Sindre, and Arne Solvberg. 1994. Understanding quality in conceptual modeling. *IEEE software* 11, 2 (1994), 42–49.

Panos Louridas and Christof Ebert. 2016. Machine Learning. *IEEE Software* 33, 5 (2016), 110–115.

Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. 2017. Extracting conceptual models from user stories with Visual Narrator. *Requirements Engineering* 22, 3 (2017), 339–358.

Mich Luisa, Franch Mariangela, and Novi Inverardi Pierluigi. 2004. Market research for requirements analysis using linguistic tools. *Requirements Engineering Journal (RE J)* 9, 1 (2004), 40–56.

Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requirements Engineering Journal (RE J)* 21, 3 (2016), 311–331.

Anas Mahmoud and Grant Williams. 2016. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering Journal (RE J)* 21, 3 (2016), 357–381.

Christopher D Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press, Cambridge, USA.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *52nd Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*. ACL, Baltimore,USA, 55–60.

Gary Miner, John Elder, and Thomas Hill. 2012. *Practical text mining and statistical analysis for non-structured text data applications*. Academic Press, Cambridge, USA.

Makoto Miwa, James Thomas, Alison OMara-Eves, and Sophia Ananiadou. 2014. Reducing systematic review workload through certainty-based screening. *Journal of biomedical informatics* 51 (2014), 242–253.

Azadeh Nikfarjam, Abeed Sarker, Karen O'Connor, Rachel Ginn, and Graciela Gonzalez. 2015. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. *Journal of the American Medical Informatics Association* 22, 3 (2015), 671–681.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)* 22, 10 (2010), 1345–1359.

Anna Perini, Angelo Susi, and Paolo Avesani. 2013. A machine learning approach to software requirements prioritization. *IEEE Transactions on Software Engineering (TSE)* 39, 4 (2013), 445–461.

Klaus Pohl. 2010. *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, Heidelberg, Germany.

Thomas Quirchmayr, Barbara Paech, Roland Kohl, and Hannes Karey. 2017. Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals. In *23rd International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ'17)*. Springer, Essen, Germany, 255–272.

Alejandro Rago, Claudia Marcos, and J Andres Diaz-Pace. 2016. Assisting requirements analysts to find latent concerns with REAssistant. *ASE J* 23, 2 (2016), 219–252.

Martin P Robillard, Walid Maalej, Robert J Walker, and Thomas Zimmermann. 2014. *Recommendation systems in software engineering*. Springer Science & Business, New York, USA.

Paige Rodeghero, Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE'17)*. IEEE, Buenos Aires,Argentina, 49–59.

Johnny Saldaña. 2015. *The coding manual for qualitative researchers* (3rd ed.). Sage Publications, London, United Kingdom.

Thomas Scholz and Stefan Conrad. 2013. Extraction of statements in news for a media response analysis. In *18th International Conference on Application of Natural Language to Information Systems (NLDB'13)*. Springer, Montpelliers, France, 1–12.

Burr Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.

Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, Honolulu, Hawaii, 1070–1079.

Carolin Strobl, James Malley, and Gerhard Tutz. 2009. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods* 14, 4 (2009), 323.

Jitendra Singh Thakur and Atul Gupta. 2016. Identifying Domain Elements from Textual Specifications. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*. ACM, Singapore, Singapore, 566–577.

Mark Utting and Bruno Legeard. 2010. *Practical model-based testing: a tools approach*. Morgan Kaufmann, Burlington, USA.

Vidhu Bhala VidyaSagar and S. Abirami. 2014. Conceptual Modeling of Natural Language Functional Requirements. *Journal of System and Software (JSS)* 88 (2014), 25–41.

Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal. 2015. Automatic generation of system test cases from use case specifications. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ICST'15)*. ACM, Graz, Austria, 385–396.

Wentao Wang, Nan Niu, Hui Liu, and Zhendong Niu. 2018. Enhancing Automated Requirements Traceability by Resolving Polysemy. In *26th International Requirements Engineering Conference (RE'18)*. IEEE, Banff, Canada, to appear.

Weka MultiSearch 2017. Multi Search - Weka package for parameter optimization. (2017). https://github.com/fracpete/multisearch-weka-package/ Last accessed: March 2018.

Jonas Winkler and Andreas Vogelsang. 2016. Automatic Classification of Requirements Based on Convolutional Neural Networks. In *24th International Requirements Engineering Conference Workshops (REW'16)*. IEEE, Beijing, China, 39–45.

Jonas Winkler and Andreas Vogelsang. 2018. Using Tools to Assist Identification of Non-requirements in Requirements Specifications–A Controlled Experiment. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18)*. Springer, Utrecht, The Netherlands, 57–71.

Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, Massachusetts, USA.

Hui Yang, Alistair Willis, Anne De Roeck, and Bashar Nuseibeh. 2010. Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the 25th IEEE/ACM international conference on Automated Software Engineering (ASE'10)*. ACM, Antwerp, Belgium, 53–62.

Zhe Yu, Nicholas A. Kraft, and Tim Menzies. 2018. Finding better active learners for faster literature reviews. *Empirical Software Engineering* 99 (2018), 1–26.

Tao Yue, Lionel Briand, and Yvan Labiche. 2011. A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering Journal (RE J)* 16, 2 (2011), 75–99.