Detecting Feature Requests of Third-Party Developers through Machine Learning: A Case Study of the SAP Community

Martin Kauschinger
Technical University of Munich
martin.kauschinger@tum.de

Niklas Vieth
SAP Deutschland SE & Co. KG
niklas.vieth@sap.com

Maximilian Schreieck
University of Innsbruck
maximilian.schreieck@uibk.ac.at

Helmut Krcmar
Technical University of Munich
helmut.krcmar@tum.de

Abstract

The elicitation of requirements is central for the development of successful software products. While traditional requirement elicitation techniques such as user interviews are highly labor-intensive, data-driven elicitation techniques promise enhanced scalability through the exploitation of new data sources like app store reviews or social media posts. For enterprise software vendors, requirements elicitation remains challenging because app store reviews are scarce and vendors have no direct access to users. Against this background, we investigate whether enterprise software vendors can elicit requirements from their sponsored developer communities through data-driven techniques. Following the design science methodology, we collected data from the SAP Community and developed a supervised machine learning classifier, which automatically detects feature requests of third-party developers. Based on a manually labeled data set of 1,500 questions, our classifier reached a high accuracy of 0.819. Our findings reveal that supervised machine learning models are an effective means for the identification of feature requests.

Keywords: Data-Driven Requirements Engineering, Enterprise Software, Machine Learning, Online Community, Platform Ecosystem

1. Introduction

The accurate elicitation of requirements is central for the development of successful software products (Chakraborty et al., 2010; Meth et al., 2015). Unfortunately, both researchers and practitioners have observed that many software development projects fail due to inaccurate or missing user requirements, which often result in significant financial losses for the development firm (Mathiassen et al., 2007; Rosenkranz et al., 2014). Specifically, the communication and interactions between the various stakeholders make the requirements elicitation process complex and hard to manage. Moreover, several studies have shown that users often lack the ability to specify their requirements correctly (Hansen & Lyytinen, 2010; Rosenkranz et al., 2014).

Requirements elicitation refers to the identification and extraction of conscious, unconscious and subconscious requirements of all involved stakeholders of a development project (Saiediana & Daleb, 2000). Although requirements elicitation has been an important topic in the information systems field for several decades (Chakraborty et al., 2010; Rosenkranz et al., 2014), its techniques have changed substantially in recent years. Previously, requirements elicitation was largely based on traditional elicitation techniques such as on-site observations, user interviews, focus groups or workshops (Byrd et al., 1992; Saiediana & Daleb, 2000). These traditional techniques require close and frequent interactions between system analysts and users, making them highly labor- and cost-intensive (Chakraborty et al., 2010). More recently, researchers and practitioners have acknowledged the rise of data-driven requirements elicitation techniques (Maalej et al., 2016; Meth et al., 2015). These data-driven techniques integrate newly available data sources such as app store reviews or social media posts into the elicitation process (e.g., Halckenhaeußer et al., 2022; Hoffmann et al., 2019; Kauschinger et al., 2021; Maalej & Nabil, 2015). Moreover, data-driven techniques often rely on new analytical technologies such as natural language processing, text mining or machine learning to leverage the full potential of these newly available data sources (Maalej et al., 2016; Meth et al., 2015).

For enterprise software, requirements elicitation remains challenging. On the one hand, enterprise software vendors' access to users has been cumbersome for decades because vendors primarily engage with their customers' IT departments, leaving them disconnected from the actual users of their software. In the past, enterprise software was also deployed on-premises, meaning that the vendors' software is licensed and running on proprietary systems of their customers. Such licensing models further increased the disconnect between vendors and users (Hoffmann et al., 2019; Schreieck et al., 2019). On the other hand, enterprise software vendors do not have access to the same data

URI: https://hdl.handle.net/10125/102747
978-0-9981331-6-4

sources for data-driven requirements elicitation as vendors of consumer software. For instance, existing research has shown that app reviews are a promising source for data-driven requirements elicitation (e.g., Carreno & Winbladh, 2013; Maalej & Nabil, 2015). Although there is a plethora of app reviews for consumer software, for example on Android's Play Store (Statista, 2022), app reviews for enterprise software are scarce. One reason for this scarcity is that enterprise software vendors only recently started to transition towards app store-centric platform ecosystems. Another reason is that enterprise software apps are far more complex so that their functionality is best assessed by technical experts rather than users. This reduces the number of potential reviewers significantly. Besides, app reviews evaluate and rate the functionality of individual apps. Hence, they more likely to contain requirements for those particular apps than for the core enterprise software system.

Another promising data source is usage data. While the tracking of user behavior has become common for many mobile apps and websites, enterprise software vendors have difficulties in collecting usage data because a large proportion of enterprise software is still running on-premises, which prohibits the collection of usage data by the vendor. For instance, in 2020, SAP still generated more revenue with on-premises solutions than with cloud solutions (SAP, 2022a). Yet, the possibilities for the collection of user data are rising as enterprise software vendors are continually transforming their on-premises products into cloud-based platform ecosystems that are running in their own data centers (Hoffmann et al., 2019; Schreieck et al., 2019).

In this paper, we investigate a new data source for data-driven requirements elicitation, which is specifically available for enterprise software vendors. We examine the potential of sponsored developer communities for data-driven requirements elicitation. While developers outside of the enterprise software domain often rely on autonomous developer communities such as Stack Overflow or Stack Exchange (Safadi et al., 2020), enterprise software vendors typically nurture their own, self-hosted developer communities such as the SAP Community, Salesforce's Trailblazer Community or ServiceNow's Now Community. The existing literature refers to such communities as sponsored developer communities because a for-profit organization, namely the enterprise software vendor, is funding and governing the activities of the community (Blohm et al., 2014; West & O'Mahony, 2008). Since these sponsored developer communities are self-hosted, enterprise software vendors have convenient access to the data that is shared in the community. Compared to vendors of consumer software, this is a major advantage because data from autonomous communities is hard to come by. Against this background, we explore whether enterprise software vendors can leverage their sponsored developer communities as an additional data source for data-driven requirements elicitation. In particular, we analyze whether it is possible to automatically detect feature requests in the questions of community members through a binary machine learning classifier. The motivation for such a classifier is that sponsored developer communities typically contain millions of posts, but only a few are relevant for the elicitation of requirements. For example, existing research has shown that a significant proportion of posts are focusing on errors or bugs (Beyer et al., 2020; Maalej & Nabil, 2015). Consequently, such a machine learning classifier has the potential of narrowing down the number of relevant questions so that requirements engineers and system analysts can evaluate the identified feature request in-depth. Therefore, we propose the following research question: *How effective is a supervised machine learning classifier in detecting feature requests in sponsored developer communities?*

To answer our research question, we collected data from the SAP Community and generated a manually labeled data set of 1,500 questions. Following the design science paradigm, we developed a supervised and binary machine learning classifier. We observed the highest prediction accuracy (0.8187) for the classifier when we extracted features with the pre-trained SBERT-Model and classified them with the Naïve Bayes algorithm. Our findings reveal that supervised machine learning models are an effective means for the identification of feature requests.

The remainder of this paper is structured as follows. In the next section, we clarify the theoretical background on requirements engineering and enterprise software. We then turn to the methodology where we explain the labeling process and our design science-oriented research approach. Thereafter, we present the results of our study before we interpret them and outline avenues for future research.

2. Theoretical Background

2.1 Requirements Engineering

The purpose of requirements engineering is the optimization of a development project toward the needs of its stakeholders. Thereby, much of the existing literature has characterized requirements engineering as a "staged sequence of activities and/or task objectives" (Chakraborty et al., 2010, p. 214). For instance, Browne and Ramesh (2002) argue that requirements engineering can be divided into three stages: information gathering,

representation, and verification. However, several researchers have noted that, in reality, requirements engineering is often chaotic and non-linear (e.g., Chakraborty et al., 2010; Davidson, 2002). We adopt this non-normative view and define requirements engineering as a "systematic and disciplined approach to the specification and management of requirements with the goal of understanding the stakeholders' desires and needs" (Glinz, 2020, p. 17). In line with this definition, a requirement can either be (1) a need of a stakeholder, (2) a capability or property that the system should have or (3) a documented representation of a need, capability or property (Glinz, 2020).

According to our non-normative view, requirements engineering comprises four types of requirements development techniques (Mathiassen et al., 2007). The first type are **requirements elicitation** techniques, which focus on the collection and consolidation of requirements from available sources (Glinz, 2020). Requirements elicitation is the main focus of our study and its techniques either follow a traditional approach (e.g., interviews or workshops) or a data-driven approach (e.g., mining app store reviews or social media data). The second type refers to **requirements prioritization** techniques, which are resource-based analyses and comparisons of elicited requirements. The goal of those techniques is to select and prioritize the most important requirements, while considering that software development is constrained by limited resources, for instance in terms of cost and time (Mathiassen et al., 2007). The third type are **requirements experimentation** techniques. Those techniques are software-centric and involve design variations of the software artifact as a means for the communication with users. Often, prototypes of the software artifact are shown to users, so that requirements can be refined and developers receive direct feedback on their prototype (Ravid & Berry, 2000). The fourth type are **requirements specification** techniques, which pertain to the documentation of explicit and agreed-upon requirements for continued development. Hence, those techniques are documentation-centric and use textual or graphical representations such as notation schemes or box structures (Mathiassen et al., 2007).

While the previous paragraph embedded requirements elicitation in the larger scope of requirements engineering, this paragraph synthesizes related work on data-driven requirements elicitation as it is the main topic of our study. In particular, we identified several papers that paved the way for our study. First, Meth et al. (2015) designed a requirements mining system and were among the first to use natural language processing for requirements elicitation. However, the authors focused on the processing of traditional data sources such as specification documents or interview transcripts. In our paper, we mine requirements in a new data source, namely sponsored developer communities. Second, Maalej and Nabil (2015) tested several machine learning models to classify app reviews into four categories: bug reports, feature requests, user experiences, and ratings. In our study, we also use machine learning to classify natural language documents, but we focus explicitly on feature requests as they are more likely to contain the requirements of developers and users. Third, Hoffmann et al. (2019) developed a conceptual model which explains how data-driven requirements elicitation can close the gap between enterprise software vendors and their users. Our paper builds upon this model and investigates a specific use case.

2.2 Enterprise Software

Over the past decades, many enterprise software vendors like Microsoft or SAP have transformed their on-premise solutions into cloud solutions. A fundamental advantage of cloud solutions is that they allow ubiquitous and convenient access to a shared pool of configurable computing resources (Basole & Park, 2019). Compared to on-premise solutions, which require additional hardware components to scale upwards, cloud solutions scale efficiently in both directions. With the shift towards cloud-based enterprise software solutions, many enterprise software vendors have transformed their product-centric software solutions into platform ecosystems, where third parties can develop complementary software applications (Foerderer et al., 2019; Schreieck et al., 2021). Such platform ecosystems typically comprise a platform owner which provides an extensible platform core, third-party developers who create complementary software applications, and end-users who use the platform and its applications (Ghazawneh & Henfridsson, 2013). Although on-premise solutions were extensible, too, more scalable platform ecosystems emerge from cloud-based solutions. For instance, while software extensions for on-premise solutions were deeply intertwined with the enterprise software system, cloud solutions provide a standardized integration layer which makes the coupling of third-party applications secure and convenient (Schreieck et al., 2019). Moreover, the resulting platform ecosystems are often similar to those in other software contexts, as the marketplaces for applications are in the limelight of the ecosystems. Examples of cloud-based enterprise software platforms are SAP's Business Technology Platform (formerly known as the SAP Cloud Platform and the SAP HANA Cloud Platform), Microsoft Azure or ServiceNow's Now Platform.

Third-party firms and their developers play an important role for the success of enterprise software vendors (Sarker et al., 2012). In fact, third-party firms often fill functional white spaces in the product portfolio of the enterprise software vendor, for instance through third-party applications. Thereby, third-party developers are responsible for the implementation of requirements, which is either done through customizations of the base enterprise software system or the development of custom applications, often called third-party applications. As enterprise software and its customization is complex, third-party developers often consult topic and product experts for guidance, for example by asking questions in sponsored developer communities (Huang et al., 2018). This is especially the case when third-party developers are uncertain about how to implement a specific requirement. The customization of enterprise software is limited to the functionalities that the base enterprise software system provides, and those are often standardized so that they fit multiple use cases and customers. Hence, enterprise software cannot comprise functionality for each and every use case. We assume that questions in sponsored online communities about missing functionalities can be interpreted as feature requests that indicate requirements of third-party developers, and ultimately of users. Sometimes, third-party developers might express their feature requests explicitly, for instance with quotes such as "it would be nice if functionality Y could be included in a future release". We define feature requests as high-level descriptions of desired functionality.

3. Methodology

The purpose of this paper is to investigate whether enterprise software vendors can leverage data from their sponsored developer communities for data-driven requirements elicitation. In the next subsections, we explain our research design, our case company and the generation of our labeled data set.

3.1 Research Design

We used a design science research approach for the development of our machine learning classifier (Hevner et al., 2004; Pfeffers et al., 2007). Design science research is concerned with the development of successful IT artifacts (Pfeffers et al., 2007) and is typically characterized as an iterative build-and-evaluate process (Hevner et al., 2004). Specifically, we followed Pfeffers et al. (2007) who describe that rigorous design science research consists of six steps: problem identification, definition of objectives, design, demonstration, evaluation and communication. In Table 1, we describe the actions that we carried out in each step.

3.2 Case Description

Our case company is the enterprise software vendor SAP. Founded in 1972, SAP has released various enterprise software products, which predominantly focus on enterprise resource planning. While products such as SAP R3 or SAP NetWeaver were on-premise

Table 1: Design science research methodology

Problem:	• For enterprise software vendors, requirements elicitation is challenging because they lack access to users. • Traditional requirements elicitation techniques are labor-intensive and do not scale. • Enterprise software vendors cannot rely on the same data sources for data-driven requirements elicitation as other software vendors (e.g., app reviews).
Objective:	• Investigate whether sponsored developer communities of enterprise software vendors are a potential source for mining software requirements. • Develop a machine learning classifier that automatically detects feature requests of third-party developers.
Design:	• Manually label an initial set of questions to validate that sponsored developer communities contain feature requests. • Select a suitable sample of 1,500 questions and recruit three SAP experts as labelers. • Perform preprocessing steps to optimally prepare the data for the classifier. • Test various feature extraction techniques. • Train the supervised machine learning model with various classification algorithms.
Demonstration:	• Provide the classifier with a set of test questions that the algorithm has not seen before. • Confirm that the classifier can automatically detect feature requests of third-party developers.
Evaluation:	• Evaluate the accuracy, precision, recall and F_1 measure of the classifier. • Identify the best-performing classifier and assess its optimization parameters. • Test and compare different combinations for preprocessing, feature extraction techniques and classification algorithms.
Communication:	• Submit the study to an academic conference.

solutions, SAP launched its first cloud-based solution in 2013, the SAP HANA Cloud Platform.

We have chosen SAP as our case company for multiple reasons. First, SAP is the largest vendor of enterprise software worldwide. Hence, we assume that SAP's ecosystem comprises a high number of third-party developers and users. Additionally, our study is more likely to have a high impact, when the results are relevant for a large ecosystem. Second, the SAP Community was launched in 2003 and since then, it has become a well-established sponsored developer community with more than three million users (SAP, 2022b). Compared to other software vendors, whose sponsored developer communities are relatively new, the SAP Community has become, and still is, an important knowledge resource for the entire SAP ecosystem (Huang et al., 2018). We thus assume that third-party developers frequently consult the SAP Community to benefit from the expertise of others. Furthermore, we also conclude that the SAP Community is a rich data source for mining software requirements. Third, SAP is a prime example of an enterprise software vendor. Although traditional enterprise software vendors such as SAP and Oracle differ to some extent from cloud-native vendors such as Salesforce and ServiceNow, our approach is transferable to all software vendors who maintain self-hosted and sponsored developer communities. Hence, our case company allows us to generalize our results to other enterprise software vendors. Fourth, SAP has launched the SAP Customer Influence Portal, which was formerly known as the SAP Idea Place (Kiron, 2012). In this portal, customers can suggest ideas for product improvements and vote on the ideas of others. We comprehend this portal as a means to collect requirements from users and third-party developers. However, this does not compromise the results of our study, because there is a substantial difference between the provisioning of such a portal and the automatic extraction of requirements in sponsored developer communities. While the Customer Influence Portal expects users and third-party developers to actively submit their improvement suggestions in the portal, sponsored developer communities can be passively mined for feature requests, for example with natural language processing techniques and machine learning models. Hence, a major advantage of the latter is that it does not require additional input or actions from users or third-party developers. Moreover, users and third-party developers might also be hesitant to submit improvement ideas due to time constraints or low chances of success. We conducted a pre-test for our study and identified that questions that contain feature requests were often answered by SAP employees who suggested submitting the idea in the SAP Customer Influence Portal. As explained next, we use this to our advantage when we selected the sample of our study. The following thread is an example of such a suggestion: https://answers.sap.com/questions/13252146/add-custom-field-to-template.html.

3.3. Data

We started our data collection process with the development of a web scraper. Specifically, our scraper extracted all existing discussion threads from the SAP Community that were posted before November 2020. In total, we collected about 2.6 million discussion threads, which comprise exactly the same amount of questions, 5.3 million answers and 4.7 million comments. For the training of our classifier, we exclusively relied on the questions as they are more likely to contain feature requests. Another reason is that our goal is to predict feature requests in newly posted questions and for those, answers and comments are not yet available. Typically, answers and comments contain solutions to technical issues and are therefore less relevant for the training of the classifier. However, as explained next, we used textual data from answers and comments to support the selection of an appropriate sample.

Supervised machine learning classifiers require a labeled data set from which the classifier can infer the outcome of a variable based on a given input. As we collected a vast amount of data, we had to reduce our data set to a sufficiently small sample that can be labeled by hand. Figure 1 provides an overview on our sample selection process. First, we investigated the statistical distributions for questions, answers, and comments. Thereby, we found that the SAP Community has gone through three major design changes. To avoid any biases that might result from these design changes, we limited our sample to questions that were posted in the most recent design of the community. Consequently, we excluded all questions that have been posted before 2016 and thereby reduced our sample to 337,000 questions. Second, we excluded all non-English questions as the classification of natural language documents is language-sensitive. Thereby, we narrowed our sample down to 285,000 questions. Third, we used an active machine learning approach to select our final sample of 1,500 questions. The central idea of active machine learning is to optimize the learning curve of a classifier by training it with the most informative cases (Dzyabura & Hauser, 2011; Hemmer et al., 2022). In particular, active machine learning is concerned with the optimal selection of data points that are given to the labelers for evaluation (Sener & Savarese, 2018). This selection is crucial because data labeling is time consuming and expensive, but it also determines the accuracy of the trained classifier. Active machine

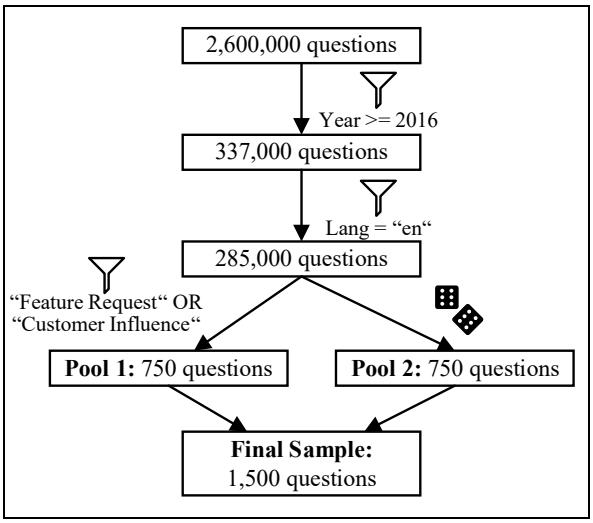


Figure 1: Sample selection

learning approaches often adopt pooling strategies which balance manually selected and informative data points with randomly selected data points (Cuong et al., 2014; Sener & Savarese, 2018). In the SAP Community, the majority of questions does not contain feature requests and hence, our classifier would perform rather inaccurate, if we train it with a random sample of 1,500 questions. Against this background, we used active machine learning and divided our remaining sample into two equally-sized pools (Batista et al., 2005; Maalej & Nabil, 2015). For the first pool, we wanted to maximize the likelihood that the questions contain feature requests. To do so, we had to identify questions that have a high probability for containing a feature request upfront. We retrieved the entire discussion threads for the existing sample and filtered for (a) threads that contain the word combination "feature request" in answers or comments and (b) for threads that mention the "Customer Influence Portal" in answers or comments. We assigned questions to the first pool if at least one filter criterion returned a hit. The logic behind this approach is to use the existing answers and comments in the community as a first assessment of whether the question contains a feature request or not. At the same time, this filtering approach does not cause any biases as the labeling and the classifier training are done on the questions exclusively. In the second pool, we considered all remaining questions which have not been identified by any of the filter criteria from pool one. Finally, we randomly selected 750 questions from each pool and ended up with an optimized but balanced final sample of 1,500 questions. Additionally, we ensure scientific rigor by using a sophisticated labeling approach which validates whether the questions actually contain a feature request or not.

We used a managed-labeler approach to label our final sample of 1,500 questions. A key reason for this approach is that the assessment of whether a question contains a feature request requires solid knowledge about SAP's products and their functionality. Hence, our labelers needed to be SAP experts to make an accurate assessment. Additionally, we wanted to have three labelers so that we can rely on the majority label when the assessments are discordant. We recruited our labelers as follows. Labeler one is part of the authors and has worked for our case company SAP for multiple years and in varying roles. Labeler two and three were recruited via the freelancer platform Fiverr. Both have a university degree and several years of experience with SAP's technology.

We used the tool Labelbox to collect the assessments from our labelers. Moreover, the labeling task was a binary evaluation, meaning that our labelers were presented with a question and they had to assess whether the question contains a feature request or not. Existing answers and comments were not shown to the labelers as our goals was to train the classifier on the questions only. The inclusion of answers and comments

Table 2: Definition of feature requests

No feature request	Feature request
• The developer is confronted with a bug or an error message and asks for help to resolve the problem. • The developer asks for a step-by-step guide to implement functionality. • The developer knows that the desired functionality is realizable through customization (for instance by reading the documentation) but does not know how.	• The developer assumes that a particular functionality is not yet implemented. • The developer does not know if something is possible but could not find information in other discussion threads or in the product documentation. • The developer has identified a function gap in the standard product. • The developer asks if a feature is included in a future release.
• **Example 1:** *"Hello Experts, please I need your help on how to calculate WIP on SAP. Steps and Tcode. Regards."* • **Example 2:** *"Hello Experts, when I am configuring SAP Hana Cloud Connector, the following error is raised. Please give me some ideas to solve this issue [IMAGE]."*	• **Example 1:** *"Hi, a customer wants to track when a user deletes or add a relationship on account level. This can't be traced in the change functionality. Is there a possibility to track this history data via another functionality or by reporting? Many thanks for your advice. Regards."* • **Example 2:** *"Hi Team, like iOS and Android Schedule setting under Server > Configuration > Schedule. Do we have anything for Windows Laptop? Regards."*

in the labeling task might have biased our results as it possible to draw conclusions from them. To ensure a common understanding for feature requests, we created a detailed labeling guide and distributed it among our labelers. The labeling guide contained a detailed definition of feature requests and several edge cases together with arguments for their assessment. Table 2 summarizes the definition of feature requests as described in the labeling guide and provides several examples. The detailed labeling guide is available from the authors upon request.

Finally, we performed several preprocessing steps to optimally prepare our data for the classifier training. First, we replaced URLs, code snippets and images with tags because machine learning classifiers have difficulties in interpreting their raw data correctly. Second, we removed remaining HTML-tags and whitespaces that resulted from the usage of our web scraper. Third, we performed the following natural language processing techniques: lowercasing, removal of punctation, numbers, possessive endings and stop-words, and lemmatization. While step one and two were necessary to reduce noise in our data, we considered step three to be optional as there are mixed empirical results on the usage of such additional preprocessing techniques (e.g., Maalej & Nabil, 2015). Hence, we tested the performance of the classifier with and without the additional preprocessing techniques of step three.

Text-based machine learning classifiers require a fixed amount of numerical features as structured input (Kowsari et al., 2019). Hence, we needed to extract these features from the unstructured text before we could train the classifier. Since we are following the design science research methodology, we tested and evaluated three common feature extraction techniques: Bag-of-Words (BoW), Term Frequency Inverse Document Frequency (TF-IDF) and document embeddings with the pre-trained language model SBERT (Kowsari et al., 2019). SBERT is a modification of the popular BERT model (Devlin et al., 2018), which benefits from a significantly reduced extraction time, while it maintains the high accuracy from BERT. SBERT has been trained on the full sentences of the Stanford Natural Language Inference corpus and the Multi-Genre Natural Language Inference corpus (Reimers & Gurevych, 2019). As any other pre-trained language model, SBERT can be fine-tuned to specific domains such as ours.

Next, we started with the model training. Specifically, we tested the performance of three binary classification algorithms which have shown promising results in similar contexts: Naïve Bayes (NB), Random Forest (RF) and Support Vector Machines (SVM) (e.g., Kühl et al., 2020; Maalej & Nabil, 2015). For all three algorithms, we used the standard threshold for binary machine learning classifiers of 0.5. Hence, the trained classifier only makes a positive prediction when the probability for a positive prediction is higher than for a negative prediction. We used Python's scikit-learn library to train the classifiers and randomly selected 1275 questions as the training set, and 225 question as the test set.

Finally, we used several measures to evaluate the performance of our classifiers. First, we use accuracy (ACC), which describes the number of correct predictions relative to the total number of predictions. Second, we use precision (P), which measures the number of correctly predicted positives relative to all positive predictions (true positives and false positives). Third, we rely on the recall rate (R) to measure the proportion of actual positives that were identified correctly. In mathematical terms, the recall rate is described as the number of true positives in relation to the number of true positives and false negatives. Finally, we calculate the F_1 measure which is the harmonic mean of recall and precision (Jiao & Du, 2016).

4. Results

In this section, we first report the results of the labeling process before we turn to the performance measures of our machine learning classifiers. Regarding the labeling, we conducted two analyses. First, we investigated the statistical distribution for each labeler and calculated a consensus label based on the assessments of our labelers. For instance, if two labelers assessed that a question contains a feature request and one labeler assessed that the question does not contain a feature request, the consensus label adopts the opinion of the majority and classifies the question as a feature request. Based on this consensus labeling approach, our final sample consisted of 606 questions with feature request and 894 questions without feature request. Figure 2 specifies the outcome of the labeling for each pool as well as for the final sample. Second, we calculated Cohen's Kappa and Fleiss' Kappa to receive a more rigorous evaluation of our labels. While Cohen's Kappa is a statistical measure for the agreement between

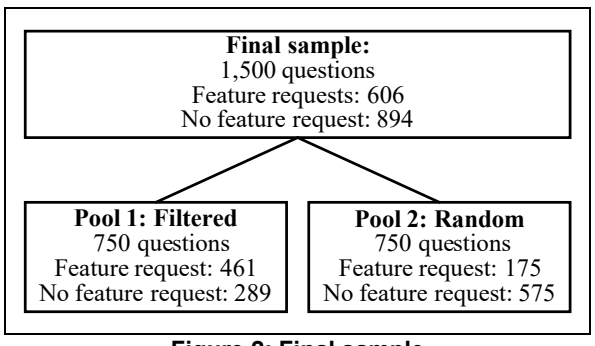
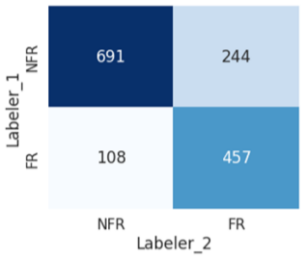
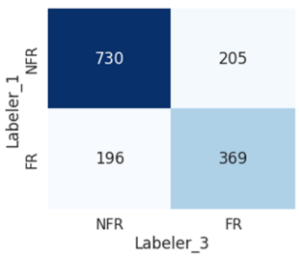


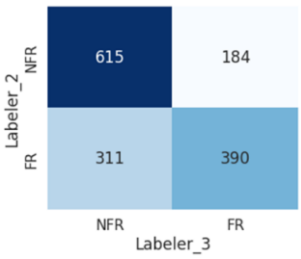
Figure 2: Final sample



Total Agreement: 76.53%
Cohen's Kappa: 0.5230
Moderate Agreement



Total Agreement: 73.27%
Cohen's Kappa: 0.4325
Moderate Agreement



Total Agreement: 67.00%
Cohen's Kappa: 0.3297
Fair Agreement

Figure 3: Cohen's Kappa

two labelers (Cohen, 1960), Fleiss' Kappa is a measure that assesses the agreement between multiple labelers (Fleiss, 1981; Powers, 2012). With a value of 0.43, the Fleiss' Kappa metric reveals an overall moderate agreement between our labelers. Additionally, we report Cohen's Kappa values for our labelers in Figure 3. We annotate feature requests as "FR" and no feature requests as "NFR". As specified in Figure 3, the agreement of our labelers ranges from 67.00 to 76.53%, resulting in a fair to moderate agreement according to Cohen's Kappa. We also investigated several questions on which our labelers disagreed in-depth. Most of these questions had room for interpretation, meaning that they could be interpreted and labeled either way. Hence, we concluded that our consensus labeling approach was a suitable means to balance this room for interpretation.

Next, we turn to the results of our classifier. As outlined in the methodology section, we investigated the influence of additional preprocessing steps (PP), feature extraction techniques (FE) and classification algorithms (CA). Hence, we trained a classifier for each possible combination of those factors. In Table 3, we report the performance metrics for these combinations. Specifically, we report the overall accuracy (ACC) of the classifier, its execution time (ET) in seconds, as well as precision (P), recall (R) and F1 measures for the feature request category. Overall, the classifiers perform reasonably well with accuracies ranging from 0.5495 to 0.8187.

Table 3: Classifier performance metrics

				FR			
PP	**FE**	**CA**	**ACC**	**P**	**R**	**F_1**	**ET**
No	BoW	NB	0.5495	0.5333	0.7912	0.6372	7
		RF	0.7912	0.7573	0.8571	0.8041	236
		SVM	0.7802	0.7802	0.7802	0.7802	346
	TF-IDF	NB	0.6374	0.6168	0.7253	0.6667	6
		RF	0.7967	0.7700	0.8462	0.8063	261
		SVM	0.7363	0.7216	0.7692	0.7447	330
	SBERT	NB	**0.8187**	0.7900	**0.8681**	**0.8272**	483
		RF	0.8132	0.7938	0.8462	0.8191	1285
		SVM	0.8022	0.7778	0.8462	0.8105	713
Yes	BoW	NB	0.6703	0.7183	0.5604	0.6296	11
		RF	0.7692	0.7692	0.7692	0.7692	252
		SVM	0.7857	**0.8095**	0.7473	0.7771	450
	TF-IDF	NB	0.6264	0.6075	0.7143	0.6566	9
		RF	0.8022	0.8090	0.7912	0.8000	266
		SVM	0.7088	0.6979	0.7363	0.7166	339
	SBERT	NB	0.7143	0.6931	0.7692	0.7292	301
		RF	0.7253	0.7030	0.7802	0.7396	1118
		SVM	0.7088	0.6827	0.7802	0.7282	539

5. Interpretation and Future Research

The purpose of this study was to investigate whether enterprise software vendors can use data from their sponsored developer communities for data-driven requirements elicitation. Specifically, we trained a supervised machine learning model for the automatic detection of feature requests of third-party developers. Overall, our study shows promising results. The best performing classifier reached a high accuracy of 0.8187 while extracting features with the pre-trained SBERT model and using the Naïve Bayes classification algorithm. This means that the classifier was able to identify 81,87% of all feature and non-feature requests correctly. We thus conclude that enterprise software vendors can use supervised machine learning classifiers to automatically detect feature requests in their sponsored developer communities. The aforementioned classifier also shows the highest recall value for the feature request category. For our study, this is a crucial measure because a high recall value indicates a minimization of false negative predictions. Assuming that an appropriate threshold value has been used, this is most desirable because a minimization of false negative predictions means that the classifier reduces the number of "missed" feature requests. In contrast, false positive predictions are more tolerable because those are questions that were falsely classified as feature request. The reason is that false positive predictions can be sorted out by a system analyst or requirements engineer in a manual review. However, the chances of correcting false negative predictions are relatively low—it is rather

unlikely that a requirements engineer will scroll through the entire community and stumble over a "missed" feature request. Hence, we see our machine learning classifier as a means to filter out relevant questions of the community and present those to system analysts and requirements engineers for further evaluation.

Besides that, all of our classifiers performed reasonably well. The accuracy of the classifiers ranges from 0.5495 to 0.8187, whereby most of them were in the 0.7 to 0.8 range. We observed the lowest accuracy when we used no additional preprocessing, feature extraction with Bag-of-Words and the Naïve Bayes classification algorithm. Apart from that, the pre-trained SBERT model performed significantly better without optional preprocessing steps (see section 3.3). However, this is not entirely surprising because the SBERT model has been trained on full sentences and preprocessing steps such as lemmatization or the removal of stop-words eliminate valuable information from the model.

Despite our promising results, our study is not without limitations. First, our best-performing classifier was not able to successfully detect all feature requests of the community. We manually investigated several incorrect predictions and found that (a) false negative assignments were more frequent for long questions with a significant amount of context descriptions, and (b) false positive assignments were more frequent for short questions with limited context descriptions. We are, however, convinced that an increase in training cases would further enhance the accuracy of the classifier. Second, we conducted a single case study which is typically associated with limited generalizability. We consider the results of our study to be reproducible because SAP and its community are a prime example for our context. Other sponsored developer communities such as ServiceNow's Now Community have a similar purpose and hence, our study is likely to be replicable with a data set from those communities.

Our study opens up two major avenues for future research. First, while we used sponsored developer communities to mine enterprise software requirements, future research can explore the elicitation of bugs to improve the maintenance of software products. Second, future research can explore different types of feature requests in sponsored developer communities.

Acknowledgements

The authors gratefully acknowledge the computational resources provided by the Leibniz Supercomputing Centre and the Rechnerbetriebsgruppe from Technical University of Munich. This work was in part funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project no. 444990299.

References

Basole, R. C., & Park, H. (2019). Interfirm Collaboration and Firm Value in Software Ecosystems: Evidence from Cloud Computing. *IEEE Transactions on Engineering Management, 66*(3), 368-380.

Batista, G. E. A. P. A., Prati, R. C., & Monard, M. C. (2005). *Balancing Strategies and Class Overlapping*. International Symposium on Intelligent Data Analysis (IDA 2005), Madrid, Spain.

Beyer, S., Macho, C., Di Penta, M., & Pinzger, M. (2020). What Kind of Questions Do Developers Ask on Stack Overflow? *Empirical Software Engineering, 25*, 2258-2301.

Blohm, I., Kahl, V., Leimeister, J. M., & Krcmar, H. (2014). Enhancing Absorptive Capacity in Open Innovation Communities. In J. M. Leimeister & B. Rajagopalan (Eds.), *Virtual Communities*. M.E. Sharpe Publisher.

Browne, G. J., & Ramesh, V. (2002). Improving Information Requirements Determination: A Cognitive Perspective. *Information & Management, 39*(8), 625-645.

Byrd, T. A., Cossick, K. L., & Zmud, R. W. (1992). A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques. *MIS Quarterly, 16*(1), 117-138.

Carreno, L. V. G., & Winbladh, K. (2013). *Analysis of User Comments: An Approach for Software Requirements Evolution*. 35th International Conference on Software Engineering (ICSE), San Francisco, USA.

Chakraborty, S., Sarker, S., & Sarker, S. (2010). An Exploration into the Process of Requirements Elicitation: A Grounded Approach. *Journal of the Association for Information Systems, 11*(4), 212-249.

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement, 20*(1), 37-46.

Cuong, N. V., Lee, W. S., & Ye, N. (2014). *Near-Optimal Adaptive Bool-Based Active Learning with General Loss*. 30th Conference on Uncertainty in Artificial Intelligence (UAI14), Quebec, Canada.

Davidson, E. J. (2002). Technology Frames and Framing: A Socio-Cognitive Investigation of Requirements Determination. *MIS Quarterly, 26*(4), 329-358.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding*. NAACL-HLT 2019, Minneapolis, USA.

Dzyabura, D., & Hauser, J. R. (2011). Active Machine Learning for Consideration Heuristics. *Marketing Science, 30*(5), 801-819.

Fleiss, J. L. (1981). *Statistical Methods for Rates and Proportions* (2. ed.). Wiley.

Foerderer, J., Kude, T., Schuetz, S. W., & Heinzl, A. (2019). Knowledge Boundaries in Enterprise Software Platform Development: Antecedents and Consquences for Platform Governance. *Information Systems Journal, 29*(1), 119-144.

Ghazawneh, A., & Henfridsson, O. (2013). Balancing Platform Control and External Contribution in Third-Party Development: The Boundary Resources Model. *Information Systems Journal, 23*(2), 173-192.

Glinz, M. (2020). *A Glossary of Requirements Engineering Terminology*. International Requirements Engineering Board. Retrieved August 31st, 2022 from https://www.ireb.org/en/cpre/cpre-glossary/.

Halckenhaeußer, A., Mann, F., Foerderer, J., & Hoffmann, P. (2022). *Comparing Platform Core Features with Third-Party Complements - Machine-Learning Evidence from Apple iOS*. 55th Hawaii International Conference on System Sciences, Virtual Conference.

Hansen, S., & Lyytinen, K. (2010). *Challenges in Contemporary Requirements Practice*. 43rd Hawaii International Conference on System Sciences, Koloa, Hawaii.

Hemmer, P., Kühl, N., & Schöffer, J. (2022). *Utilizing Active Machine Learning for Quality Assurance: A Case Study of Virtual Car Rendering in the Automotive Industry*. 55th Hawaii International Conference on System Sciences, Virtual Conference.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75-105.

Hoffmann, P., Mateja, D., Spohrer, K., & Heinzl, A. (2019). *Bridging the Vendor-User Gap in Enterprise Cloud Software Development through Data-Driven Requirements Engineering*. 40th International Conference on Information Systems, Munich, Germany.

Huang, P., Tafti, A., & Sunil, M. (2018). Platform Sponsor Investments and User Contributions in Knowledge Communities: The Role of Knowledge Seeding. *MIS Quarterly, 42*(1), 213-240.

Jiao, Y., & Du, P. (2016). Performance Measure in Evaluating Machine Learning Based Bioinformatics Predictors for Classifications. *Quantitative Biology, 4*(4), 320-330.

Kauschinger, M., Schreieck, M., Boehm, M., & Krcmar, H. (2021). *Knowledge Sharing in Digital Platform Ecosystems - A Textual Analysis of SAP's Developer Community*. 16th International Conference on Wirtschaftsinformatik, Virtual Conference.

Kiron, D. (2012). *SAP: Using Social Media for Building, Selling and Supporting*. Retrieved August 31st, 2022 from https://sloanreview.mit.edu/article/sap-using-social-media-for-building-selling-and-supporting/.

Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information, 10*(4).

Kühl, N., Mühlthaler, M., & Goutier, M. (2020). Supporting Customer-Oriented Marketing with Artificial Intelligence: Automatically Quantifying Customer Needs from Social Media. *Electronic Markets, 30*, 351-367.

Maalej, W., & Nabil, H. (2015). *Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews*. 23rd International Requirements Engineering Conference, Ottawa, Canada.

Maalej, W., Nayebi, M., Johann, T., & Ruhe, G. (2016). Toward Data-Driven Requirements Engineering. *IEEE Software, 33*(1), 48-54.

Mathiassen, L., Tuunanen, T., Saarinen, T., & Rossi, M. (2007). A Contingency Model for Requirements Development. *Journal of the Association for Information Systems, 11*(2), 569-597.

Meth, H., Mueller, B., & Maedche, A. (2015). Designing a Requirement Mining System. *Journal of the Association for Information Systems, 16*(9), 799-837.

Pfeffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems, 24*(3), 45-77.

Powers, D. M. W. (2012). *The Problem with Kappa*. 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France.

Ravid, A., & Berry, D. M. (2000). A Method for Extracting and Stating Software Requirements that a User Interface Prototype Contains. *Requirements Engineering, 5*(4), 225-241.

Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. EMNLP 2019, Hong Kong, China.

Rosenkranz, C., Vranešić, H., & Holten, R. (2014). Boundary Interactions and Motors of Change in Requirements Elicitation: A Dynamic Perspective on Knowledge Sharing. *Journal of the Association for Information Systems, 15*(6), 306-345.

Safadi, H., Johnson, S. L., & Faraj, S. (2020). Core-Periphery Tension in Online Innovation Communities. *Organization Science, Preprint*.

Saiediana, H., & Daleb, R. (2000). Requirements Engineering: Making the Connection Between the Software Developer and Customer. *Information and Software Technology, 42*, 419-428.

SAP. (2022a). *2021 Q4 Quarterly Statement*. Retrieved August 31st, 2022 from https://tinyurl.com/3bfp8cef.

SAP. (2022b). *Welcome to the SAP Community*. Retrieved August 31st, 2022 from https://community.sap.com/.

Sarker, S., Sarker, S., Sahaym, A., & Bjorn-Andersen, N. (2012). Exploring Value Cocreation in Relationships Between an ERP Vendor and its Partners: A Revelatory Case Study. *MIS Quarterly, 36*(1), 317-338.

Schreieck, M., Wiesche, M., & Krcmar, H. (2021). Capabilities for Value Co-creation and Value Capture in Emergent Platform Ecosystems: A Longitudinal Case Study of SAP's Cloud Platform. *Journal of Information Technology, 36*(4), 365-390.

Schreieck, M., Wiesche, M., Kude, T., & Krcmar, H. (2019). *Shifting to the Cloud - How SAP's Partners Cope with the Change*. 52nd Hawaii International Conference on System Sciences, Grand Wailea, Hawaii.

Sener, O., & Savarese, S. (2018). *Active Learning for Convolutional Neural Networks: A Core-Set Approach*. International Conference on Learning Representations (ICLR 2018), Vancouver, Canada.

Statista. (2022). *Average Rating and Number of Reviews in the Google Play Store*. Retrieved August 31st, 2022 from https://www.statista.com/statistics/1296490/ratings-and-reviews-android-apps-by-category/.

West, J., & O'Mahony, S. (2008). The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation, 15*(2), 145-168.