2016 IEEE 24th International Requirements Engineering Conference

Software Feature Request Detection in Issue Tracking Systems

Thorsten Merten*, Matúš Falis†, Paul Hübner‡, Thomas Quirchmayr‡, Simone Bürsner* and Barbara Paech‡
*Bonn-Rhein-Sieg University of Applied Sciences
Dept. of Computer Science Sankt Augustin, Germany Email: {thorsten.merten,simone.buersner}@h-brs.de
†The University of Edinburgh, Edinburgh, United Kingdom, Email: matfalis@gmail.com
‡University of Heidelberg, Software Engineering Group, Heidelberg, Germany,
Email: {huebner, thomas.quirchmayr, paech}@informatik.uni-heidelberg.de

***Abstract*—Communication about requirements is often handled in issue tracking systems, especially in a distributed setting. As issue tracking systems also contain bug reports or programming tasks, the software feature requests of the users are often difficult to identify. This paper investigates natural language processing and machine learning features to detect software feature requests in natural language data of issue tracking systems. It compares traditional linguistic machine learning features, such as "bag of words", with more advanced features, such as subject-action-object, and evaluates combinations of machine learning features derived from the natural language and features taken from the issue tracking system meta-data. Our investigation shows that some combinations of machine learning features derived from natural language and the issue tracking system meta-data outperform traditional approaches. We show that issues or data fields (e.g. descriptions or comments), which contain software feature requests, can be identified reasonably well, but hardly the exact sentence. Finally, we show that the choice of machine learning algorithms should depend on the goal, e.g. maximization of the detection rate or balance between detection rate and precision. In addition, the paper contributes a double coded gold standard and an open-source implementation to further pursue this topic.**

I. Introduction

Software feature requests (SFR) play an important role in requirements engineering and the whole software engineering process. SFRs are usually communicated via an issue tracking system (ITS) by customers or users of the software and are further processed by project members. Therefore, the ITS brings multiple stakeholder groups together [1] and holds multiple SFRs [2], [3]. Knowing all SFRs is helpful for common software engineering tasks, such as checking whether all feature requests are implemented, updating the documentation for implemented feature requests, or even billing (did we add new features to the software while this bug was being resolved?).

Ideally, an SFR is formulated clearly and precisely and its respective issue is flagged as feature. In this way the SFR can be distinguished from other information in the ITS, such as bug reports or development tasks. In practice, however, finding SFRs may not be straightforward, as in the following two situations: (1) The issue that includes the SFR is categorized wrongly [4] (e.g. as a bug instead of a feature) or not at all [3]. As a consequence, the SFR may not be explicitly visible in the ITS and is thus hard to find. (2) The SFR is hidden in a comment to the issue, e.g. because the feature idea emerged during bug resolution [3]. The SFRs in both examples are hard to detect, especially if numerous issues are analyzed retrospectively.

The main goal of this paper is to study whether SFRs can be detected automatically to cope with such problems. We evaluate multiple machine learning (ML) algorithms, different text preprocessing techniques, and Machine Learning Features (MLFs) to detect the following three parts of an SFR [3]: (1) "requests", text that requests for a distinguishing characteristic of a software item (e.g. a quality or functionality) that provides value for users of the software; (2) "clarifications", text that explains a request, e.g. because the functionality needs further context; and (3) "solution proposals", text that describes implementation ideas for an SFR.

Challenges in classifying ITS natural language (NL) data are introduced in Section II along with a brief background on ML, text pre-processing, and ITSs. Related literature can be found in Section III and is split in related feature extraction work and literature which serves as a basis for some of our MLFs.

The main research goal is divided into three research questions in Section IV and the experiment setup is described thereafter in Section V. Our main findings are that request can be detected best out of the researched SFR parts. This detection works best if linguistic MLFs are used in combination with MLFs from ITS meta-data. Our evaluation shows that issues or data fields containing SFRs can be identified reasonably well, but it is rather hard to identify the exact sentences describing the SFRs. Finally, the choice of ML algorithms should depend on the corresponding data mining goal: some algorithms maximize the detection rate of SFRs, others maximize the balance between detection rate and precision. The results are presented in detail in Section VI and discussed in Section VII. The final two sections discuss study related threats to validity and conclude the paper.

II. Background

This section briefly introduces ML in general and how ML performance can be measured. Then it gives an overview of ITS and details challenges of ITS data in the context of ML.

2332-6441/16 $31.00 © 2016 IEEE
DOI 10.1109/RE.2016.8

RE 2016, Beijing, China
Research Paper

A. Machine Learning Background

In the experiments supervised ML algorithms were employed. Supervised learning means that previously labeled training data is used for training the model [5], [6]. For the purpose of training, a set of MLFs and their corresponding labels are needed. For example, a model is trained to predict whether an issue should be classified as *feature* or *bug* using *all the words that occur in the issue*. In this case *feature* and *bug* are the labels and *all the words that occur in the issue* is the MLF. This MLF is also called *Bag Of Words* (BOW).

An ML algorithm calculates a statistical prediction model from MLFs. In general, the prediction model is more precise, the more examples (training data) are provided. Besides the amount of training data, the selection of MLFs is very important [7], [8]. As an example, assume the ML model from the example above was not trained with the BOW, but with the number of comments to each issue. Then, it would be difficult for the ML algorithm to compute a precise prediction model[1]. However, the combination of both BOW and number of comments may improve the results.

In this paper, the following 7 algorithms that are often employed in text classification tasks, are used to detect SFRs [9]: (1) Naive bayes (NB); (2) Multinomial naive bayes (MNB); (3) Linear support vector classifier (also known as support vector machine, SVM); (4) Logistic regression (LR, also known as maximum-entropy); (5) Stochastic gradient descent (SGD); (6) Decision tree (DT); (7) Random forest (RF); We use the default settings, as provided by the NLTK [10] and the Scikit-learn [9] APIs, for all our experiments, which implies that no parameter tuning was done. Details on ML algorithms can be found in [5], [10], or [6].

B. Issue Tracking System Data

ITSs are utilized in many projects [11] to store SFRs, bug reports, and to discuss and keep track of the corresponding development tasks. Thus, the data in ITSs comprise manifold topics [3].

An excerpt of an SFR, taken from our data, is shown in Figure 1. An issue comprises at least two separate data fields: a title (or summary) and a description. Additionally issues can be commented in almost every ITS, as shown in Figure 1 below of "History". We refer to title, description, and comments as *ITS data fields* in the remainder of this paper. Each *data field* comprises two parts: NL text and *data field meta-data*, e.g. timestamp, author, etc. The issue itself includes *ITS meta-data*, too, e.g. the status or the *issue type*. ITSs support either defining one issue type per issue, or tagging issues with multiple descriptors.

C. Impact of ITS NL data on ML algorithms

There are three major problems in ITS NL data, impeding SFR detection: (1) Flaws in the natural language, (2) mixing natural language with technical noise, and (3) flaws in the assignment of issue types.

[1]However, such a classification is in rare cases possible, e.g. if every *bug* issue has more comments than a *feature* issue or vice versa.

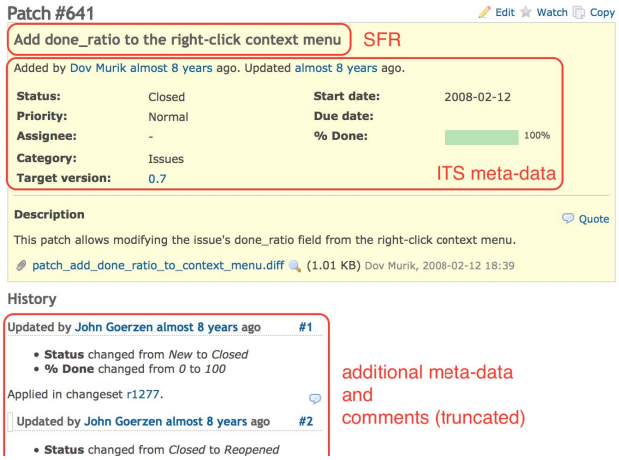

Fig. 1. Screenshot of Redmine Issue 641 with Manual Annotations

Flaws in the natural language: In previous research [3] we studied the NL content in ITS data. We found that NL is often used imprecisely, contains flaws, or does not form a whole sentence. Issues are seldom corrected or re-organized, and some issues or comments represent only hastily written developer notes. In addition, the comments to an issue often drift from the original topic, mentioned in the title and the description, towards something different (e.g. a discussion about a new feature in an issue that initially describes a bug). Since ML algorithms use properties of the NL to learn their models, spelling errors, hastily written notes, and content drifts have a negative impact on the performance.

Mixing NL with technical noise: NL is mixed with technical noise such as source code, stack traces, or links. Sometimes this noise can easily be filtered but often technical artifacts, such as class names or HTML-tags, occur as parts of a sentence. Such technical artifacts can have a positive or negative impact on ML algorithms: On the one hand, an ML algorithm could learn that SFRs typically do not contain patterns of a stack trace. On the other hand, code snippets, such as package names, might confuse the algorithm because they do not follow the same patterns as NL.

Flaws in the assignment of issue types: Issue types, such as "feature" or "feature request", should actually denote issues containing SFRs. In practice, however, issue types are often not available at all [3] and if appropriate issue types are used, they are often assigned incorrectly [4].

D. Levels of Detail for ITS ML Evaluation

Issues contain data fields, data fields contain NL text, and NL text is composed of sentences[2]. Hence, the detection of SFRs can be approached on different levels of detail. Detection on *issue* level reveals, whether the issue contains one or many SFRs. Detection on *data field* level reveals, whether the title, the description, or a certain comment to the issue contains one or many SFRs. Detection on *sentence* level reveals, whether

[2]We use sentence as synonym for phrases, bullet points, and so on.

a certain sentence in the NL describes an SFR. In figures and tables we will refer to these levels as *I*, *DF*, and *Se*.

In general terms: The higher the level of detail, the harder is a detection due to the following two circumstances:

- On a lower level, more MLFs can be employed to train the model. E.g. on issue level, the words in the title, description and comments can be used to create the BOW MLF. In contrast, on sentence level only the words of that sentence form the BOW.
- More objects need to be classified on a higher level with the same amount of training data. E.g our data set includes > 200 SFR annotations. These are used for a detection within 599 issues and 11149 sentences, respectively.

For each level, some MLFs can be derived directly (e.g. issue meta-data MLFs on issue level). To gather the MLFs from other levels, we employ the concepts of inheritance and aggregation. E.g. when the detection is done on sentence level, each sentence can inherit the meta-data MLFs from the comprising data field and issue. If the detection is done on issue level, each issue accumulates the MLFs from its embedded data fields and, transitively, the MLFs derived from their textual contents. On data field level both concepts are used. This allows to compare the same combinations of MLFs on different levels and thus evaluate which level yields the best results.

E. Measuring Machine Learning Performance

ML performance is typically measured by comparing the results of a classifier to a manually created gold standard.

An ML algorithm classifies data (e.g., text document) along predefined labels (e.g. SFR). More precisely, the trained model determines whether a piece of data (e.g., a sentence) corresponds to predefined MLFs of a label or not (e.g., either the data is an SFR or not). This classification can then be either correct or not. Thus, a classification ends up in four different measures of relevance: (1) the piece of data is correctly classified to belong to the label (true positive, TP), (2) the piece of data is correctly identified not to belong to the label (true negative, TN), (3) the piece of data is wrongly classified to belong to the label (false positive, FP), (4) the piece of data is wrongly identified not to belong to the label (false negative, FN). Based on these four measures of relevance, recall (R) and precision (P) are defined:

$$R = \frac{TP}{TP + FN} \qquad P = \frac{TP}{TP + FP} \tag{1}$$

R measures how many relevant SFRs are found and P measures how many relevant SFRs are found correctly. Since R can be maximized by classifying all sentences as SFRs, which results in a low P and P can be maximised by classifying a single SFR correctly, which results in low R, the F_β-Measure, as their harmonic mean, is often employed. In the paper we report on P, R and the F_1 measure, which balances P and R:

$$F_\beta = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P) + R} \tag{2}$$

In addition, we report on a measure denoted as $\text{MAX}(R), P_{\geq p}$ throughout the paper as it is often desired to maximize recall in favor of precision as argued e.g. by Berry [12]. $\text{MAX}(R), P_{\geq p}$ is defined as best achievable recall if the precision is higher than $p\%$. In other words, a reasonable precision is defined to be at least $p\%$. The following example illustrates the use of $\text{MAX}(R), P_{\geq p}$: With 76 SFRs, as in the agreed upon data set, a $\text{MAX}(R), P_{\geq 0.05}$ yields $76 \times 20 = 1,444$ hits. This implies that $1,444 - 76 = 1,364$ false positives have to be winnowed manually, to correctly identify all SFRs ($P = 1$). Hence, the additional manual work is amortized, if the number of objects to be classified is $> 1,364$ for 76 SFRs.

This is the case on the data field level, which consists of 599 titles $+599$ descriptions $+3519$ comments $= 4717$ data fields, and on the sentence level, which consists of 11149 sentences, as shown in Table I. $\text{MAX}(R), P_{\geq p}$ will be reported for $p = 0.05$ on the sentence and data field levels, and $p = 0.2$ on the issue level[3].

III. Related Work

First, this section presents mining approaches for SFR. Then, it summarizes related work used to define our MLFs.

A. Mining Approaches to Identify Software Features

Bakar et al. [13] performed a systematic literature review on feature extraction approaches from natural language requirements in context of software product lines. Their review includes multiple methods to summarize, cluster, or abstract from software requirements to generate a representation of software features. In contrast to the reviewed approaches, we use a bottom-up technique to detect NL describing parts of software features (e.g. feature request or feature clarification). In addition, we use ITS data as input whereas the reviewed approaches in [13] utilize requirement documents or product descriptions, which are usually better structured.

On ITS side, much work is related to classification of issues, e.g. bug vs. feature [14], [15] with improving results. Our work is complementary, and we report on the ability to identify SFRs in ITSs on different levels of detail.

Vlas et al. [16] present a bottom-up approach similar to ours. They use heuristics and language engineering to detect software requirements in ITSs. They assume that a software requirement is described by a single sentence and detect the requirements by means of language patterns and keyword lists. In contrast, we use supervised ML algorithms since we cannot ensure SFRs to be described by a single sentence.

[3]With these settings for p sound values for R can be achieved. If p is doubled ($p \in 10, 20$), the results for R become significantly worse.

B. MLF Suggestions to Identify Software Features

Guzman et al. [17] use collocations (also called bi-grams [10]) to denote fine grained software features in app store reviews. They employ a likelihood-ratio test [18] to filter likely irrelevant collocations. We adopt their idea and create MLFs out of bi-grams and tri-grams. Fitzgerald et al. [19] use a mixture of ITS meta-data MLFs and linguistic MLFs for early failure prediction in ITS. Maalej et al. [20] show that meta-data, such as star ratings in app stores, can improve SE-related classification tasks. Based on the ideas in [19], [20], we derive MLFs from *ITS meta-data* and *data field meta-data*. Moreover, we add the concepts of inheritance and aggregation to make use of this meta-data on different levels.

Besides other heuristics, the aforementioned approach in [16] uses a subject-action-object (SAO) pattern based on [21] as central basis for their heuristics. The action is actually the main verb of the sentence filtered by different heuristics and keyword lists, e.g. "need" in "The software **needs** logging". Though a replication of all their heuristics is unfeasible in the ML context, we adopt their central approach and derive MLFs from subject-verb-object triplets based on typed dependency parsing [22].

IV. Research Questions

The overall research question of this paper, whether software feature requests can be detected automatically in ITS NL data, is divided in the following three research questions:

RQ1 How should text be preprocessed for SFR detection?

RQ2 Which combinations of MLFs derived from the NL and ITS meta-data should be used for SFRs detection on different levels of detail?

RQ3 How well can trained prediction models be reused?

Section II-C states problems with ML and ITS data, that can often be mitigated with preprocessing techniques. RQ1 discusses which combination of such techniques should be used. RQ2 contrasts rather traditional MLFs with ideas from related work and takes MLFs based on issue and data field meta-data into account. Furthermore, it investigates whether the very sentence that contains the SFR can be extracted or if more data (e.g. from the comprising data field or issues) is needed. RQ3 asks whether models have to be trained for every project or if trained models can be applied to other projects, different from the projects used for training.

V. Research Process

The research process has four important steps: (1) The extraction and preparation of the data, (2) the creation of a gold standard, (3) the engineering of MLFs, and (4) the evaluation of different ML algorithms, MLFs, and preprocessing techniques.

A. Data Extraction

150 issues were randomly extracted out of the first 1000 issues[4] that were committed to each of the following four open-source projects:

[4]Older issues often contain more SFRs than newer issues [2].

TABLE I
Project Characteristics

	c:geo	Lighttpd	Radiant	Redmine	all
Software Type	Android app	HTTP server	CMS	ITS	
Audience	consumer	technician	consumer developer	developer	
Programming Lang.	Java	C	Ruby	Ruby	
ITS	GitHub	Redmine	GitHub	Redmine	
ITS Usage	ad-hoc	structured	ad-hoc	structured	
ITS size (# of issues)	3850	2900	320	19.000	
Open issues	450	500	50	4500	
Closed issues	3400	2400	270	14.500	
Project size (in LOC)	130,000	41,000	33,000	150,000	
Extracted Issues	150	150	150	149	599
Marked as Feature	12	39	0	73	124
Marked as Bug	23	111	6	52	192
Marked as Other	47	0	36	24	107
Without Issue Type	68	0	108	0	176
# of Comments	1060	569	658	1232	3519
Avg. # Comm. / Issue	7.1	3.8	4.4	8.3	5.9
# Sentences	2896	1975	2044	4234	11149
LOC	181	1266	217	526	2190

- *c:geo*, an Android application to play a real world treasure hunting game.
- *Lighttpd*, a lightweight web server application.
- *Radiant*, a modular content management system.
- *Redmine*, an ITS.

The projects show different characteristics with respect to software type, intended audience, programming languages and used ITS. Details of these characteristics and the amount of extracted data is shown in Table I separated by a horizontal line. The c:geo and the Radiant projects use the GitHub ITS whereas Redmine and Lighttpd use the Redmine ITS. Therefore, the issues of the first two projects are categorized by tagging and each issue of the other two projects is assigned to exactly one category, e.g. feature or bug as in Table I. c:geo was chosen because it is an Android application, and thus the ITS contains more consumer requests than the other projects. Lighttpd was chosen because its ITS contains more code snippets and noise than the other projects. Radiant was chosen because its issues are not categorized as feature or bug at all and it contains fewer issues than the other projects. Finally, Redmine was chosen because it is a very mature project and ITS usage is very structured, compared to the other projects. Some of the researchers were already familiar with these projects since we reported on ITS NL contents earlier [3].

B. Gold Standard

We manually created a gold standard to train the models and evaluate their results. Each sentence of each selected issue (including all comments) was annotated – if relevant – by two different persons (annotators) independently with one of the following labels[5]:

- Request Functionality *to denote functional SFRs*.
- Request Quality[6] *to denote quality SFRs*
- Solution *to denote technical solutions for an SFR*.

[5]Among others, which are irrelevant for this paper.

[6]Too little data was found to train a model for requests for quality.

TABLE II
EXTRACTED DATA AND ANNOTATIONS

	# Annotations / Avg. # of Sentences per Annotation **Annotations agreed upon by two coders.**				
Label	c:geo	Lighttpd	Radiant	Redmine	all
Req. Functionality	23 / 1.08	9 / 1.0	10 / 1.0	34 / 1.12	76 / 1.05
Request Quality	4 / 1.0	0	0	0	4 / 1.0
Solution	18 / 1.0	2 / 1.0	6 / 1.16	10 / 1.0	36 / 1.04
Clarification	46 / 1.04	17 / 1.12	11 / 1.27	26 / 1.23	100 / 1.17
	Uncertain cases.				
Req. Functionality	70 / 1.17	28 / 1.0	43 / 1.07	155 / 1.21	296 / 1.11
Request Quality	8 / 1.0	1 / 1.0	4 / 1.0	4 / 1.0	17 / 1.0
Solution	72 / 1.36	14 /1.5	28 / 1.36	148 / 1.41	262 / 1.41
Clarification	223 / 1.50	77 / 1.44	110 / 1.21	234 / 1.38	644 / 1.38

- Clarification *to denote additional explanations to an SFR.*

A guidebook with examples was created and multiple issues were discussed in the group, to gain a common understanding of the annotations. The annotators classified the sentences based on the definition of the labels given in Section I. In our results we report on two data sets that are composed of the very same issues: (1) the sentences that were annotated identically by two annotators, as the *agreed upon cases* data set, (2) and the sentences that were only found by a single annotator, as the *uncertain cases* data set. Both data sets are summarized in Table II. On average, the annotators agreed with a Cohens kappa [23] of 0.91 on the labels for issue titles and 0.88 on the labels for issue descriptions, which is a rather high agreement. Due to limitations in our annotation tool, we cannot report on the exact kappa for the issue comments. However, we observed significantly lower agreements in random samples of comments, compared to title and description. This can also be seen in Table II which shows a disparity of labels for the two data sets. These differences imply that even human experts have a tendency not to agree upon whether a sentence contains an *SFR* , a *request for quality* , an *clarification*, or a *solution*. The influence of agreement factors will be discussed in Section VIII.

During the annotation phase, the annotators collected keywords for the *Request Functionality* label. New keywords were added to a keyword list, if a word was noticed repeatedly by the annotator, whereas "repeatedly" was generally interpreted as "more often than $5 - 15$ times". Examples from this collection are modal verbs such as *should*, *would* and *could* as in "we **should** implement <functionality>" or "**could** you please add <functionality>". The full keyword list is used as one MLF in the detection, as described in the following Section.

C. Machine Learning Features

To train a classifier, different MLFs, all modeled as binary features, were employed. MLFs that are not binary by nature, such as the number of comments of an issue, are quantized. The following MLFs were used to conduct the experiments:

BOW: As reference for other MLFs in the experiment, the BOW is employed (see Section II-A).

Bi- and Tri-Grams: We expand the BOW with bi- and tri-grams in our measurements. Both, bi- and tri-grams denote words that occur together in tuples or triplets. Due to the fact that a huge amount of bi- and tri-grams can occur in the textual content of a single issue, the number of extracted bi- and tri-grams is restricted to 200, as rated by a chi-square-test [10].

SAO: To replicate the SAO pattern from [16], the Stanford Dependency Parser [22] is employed. The Stanford Dependency Parser outputs a directed acyclic graph of all words in a sentence. If a sentence contains a main verb, we check for each subject and object whether a path can be found between subject, main verb and object. If and only if (1) a sentence contains a main verb, and (2) at least one subject and object, and (3) the subject(s) and object(s) are indirectly connected to the verb, these triplets are used as MLFs.

Keywords: The keyword list, gathered during data annotation (see Section V-B), is used to check whether a simple gathering technique improves the ML model. In addition, a keyword list, which contains positive words, is employed since SFRs tend to be written gently and politely [3].

Issue Meta-Data: In addition to the linguistic MLFs described above, MLFs derived from issue meta-data are considered in the experiments: (1) the issue type (or the tags, if appropriate, as described in Section II-B), (2) the users that participated in the issue (the author and all users that commented or changed the issue), (3) the author of the issue, (4) the duration of the issue[7], (5) the number of comments to the issue[8], (6) all possible former states of the issue (e.g. open, closed, in development, in test, ...), and (7) the current status (open, in progress, closed, etc.).

Data Field Meta-Data: The following meta-data was extracted from ITS data fields: (1) the data field author, (2) whether the issue status changed while the data field was updated, (3) the duration since the previous comment.

D. Text Preprocessing

Five techniques are employed to preprocess the NL texts in the experiments:

1) *Lowercasing*: Each letter is lowercased, e.g. "Feature" or "featURE" become "feature". The technique helps to treat capitalized words in the same way as all the other words. Since we are not interested in names or places, lowercasing should not bear any disadvantages.

2) *Stemming* [24]: Stemming is the process of reducing words to their word-stem. E.g. the words: "implementing" and "implemented" become "implement" after stemming. This is a useful technique to equalize verbs with different tenses unless the verbs are irregular.

3) *Punctuation removal*: The following characters are removed: `'.', ',', '?', '!', ':', ';', '-', '(', ')', '[', ']', "'", '"', '/'`.

4) *Separation of technical noise*: Code and stack-traces are separated, before the NL is processed. To separate technical noise from NL, the text denoted by special code tags (e.g. `<pre>...</pre>` in Redmine or a special indentation

[7]Quantized as < 4 hours, ≤ 8 hours, ≤ 1 day, ≤ 10 days or > 10 days

[8]Quantized as < 1, ≤ 2, ≤ 4, ≤ 6, > 6; quantization was made on the basis of [3, Fig. 2]

TABLE III
EVALUATED PREPROCESSING TECHNIQUES AND LINGUISTIC MLF-SETS

Setting	Lower	Stem	Punc-tuation	Sep. Tech. Data	Stop-words
$\bar{1}$	-	-	-	-	-
$\bar{2}$	✓	✓	-	-	-
$\bar{3}$	✓	✓	-	-	✓
$\bar{4}$	✓	✓	-	✓	-
$\bar{5}$	✓	✓	-	✓	✓
$\bar{6}$	✓	✓	✓	-	-
$\bar{7}$	✓	✓	✓	-	✓
$\bar{8}$	✓	✓	✓	✓	-
$\bar{9}$	✓	✓	✓	✓	✓

(a) Preprocessing Settings

MLF Set	BOW	bi- & tri-grams	SAO
$\ddot{1}$	✓	-	-
$\ddot{2}$	-	✓	-
$\ddot{3}$	-	-	✓
$\ddot{4}$	✓	✓	-
$\ddot{5}$	✓	-	✓
$\ddot{6}$	✓	✓	✓
$\ddot{7}$	-	✓	✓
$\ddot{8}$	✓	-	✓

(b) Linguistic MLF-Sets

in Github) is removed. Although more advanced techniques can be employed, e.g. [25], a careful inspection of the data revealed that a removal by code tags already yields good results on our data samples.

5) *Stop-word removal*: Common English words that usually do not carry much semantic meaning are removed, using the stop-word list distributed with NLTK toolkit [10]. However, as described in Section V-C, many modal verbs are in our keyword lists. Hence, only the stop-words not in the keyword lists are removed, whenever the keyword MLF is active.

E. Evaluation

Each experiment is run with all of the algorithms introduced at the end of Section II-A.

Data Splitting and Measuring: To evaluate the results, a classical ten-fold-cross validation is employed. First, the issues are randomized and divided into 10 equal groups. Each experimental run then uses the issues of 9 groups for training. The issues of the remaining group serve as evaluation data. Each experiment runs 10 times, thus utilizing each issue 9 times as training data and one time as evaluation data. To answer RQ3 a different setup is necessary. The selected issues of three projects are used for training and the issues of the fourth project are used for evaluation. This results in a $75\%/25\%$ split.

VI. RESULTS

All results presented in the following sections are available for download[9]. To answer RQ1, the performance among different ML algorithms with different preprocessing settings on linguistic MLFs is compared. Then, the best performing preprocessing setting is used in further evaluations to answer RQ2 and RQ3[10].

A. Best Preprocessing Techniques (RQ1)

As the combination of all preprocessing techniques results in 2^5 combinations, the reporting is constrained on the preprocessing settings defined in Table III (a): the first setting uses no preprocessing as a reference. Lowering and stemming are considered best practices [18], [26] and are therefore included in all further preprocessing settings. All combinations of punctuation removal, stop-word removal, and separation of technical data are evaluated. These preprocessing techniques influence all the linguistic MLFs derived from the NL text, hence the preprocessing results are reported for all linguistic MLF-sets shown in Table III (b).

TABLE IV
AVERAGE AND MAXIMUM F_1 SCORES FOR PREPROCESSING

		Max F_1, BOW Only			Avg	Max F_1, All MLF Sets			Avg
Agreed Cases									
Level	Label	Conf.	Alg.	Value	F_1	Conf.	Alg.	Value	F_1
I	Req. Funct.	$\bar{6}$ / $\ddot{1}$	LR	0.495	0.279	$\bar{3}$ / $\ddot{6}$	SGD	0.643	0.168
	Clarification	$\bar{3}$ / $\ddot{1}$	SGD	0.401	0.201	$\bar{3}$ / $\ddot{1}$	SGD	0.401	0.124
	Solution	$\bar{3}$ / $\ddot{1}$	MNB	0.343	0.097	$\bar{2}$ / $\ddot{6}$	MNB	0.492	0.062
DF	Req. Funct.	$\bar{7}$ / $\ddot{1}$	MNB	0.130	0.088	$\bar{3}$ / $\ddot{6}$	MNB	0.195	0.054
	Clarification	$\bar{8}$ / $\ddot{1}$	SGD	0.102	0.089	$\bar{3}$ / $\ddot{6}$	MNB	0.119	0.055
	Solution	$\bar{7}$ / $\ddot{1}$	MNB	0.104	0.036	$\bar{6}$ / $\ddot{6}$	MNB	0.109	0.022
Se	Req. Funct.	$\bar{7}$ / $\ddot{1}$	MNB	0.078	0.048	$\bar{9}$ / $\ddot{6}$	MNB	0.106	0.030
	Clarification	$\bar{0}$ / $\ddot{1}$	SVN	0.055	0.047	$\bar{9}$ / $\ddot{5}$	MNB	0.078	0.029
	Solution	$\bar{7}$ / $\ddot{1}$	MNB	0.059	0.02	$\bar{6}$ / $\ddot{6}$	MNB	0.088	0.013
Uncertain Cases									
I	Req. Funct.	$\bar{7}$ / $\ddot{1}$	LR	0.752	0.501	$\bar{7}$ / $\ddot{4}$	LR	0.757	0.314
	Clarification	$\bar{7}$ / $\ddot{1}$	LR	0.551	0.412	$\bar{7}$ / $\ddot{4}$	LR	0.556	0.304
	Solution	$\bar{9}$ / $\ddot{1}$	SVM	0.498	0.258	$\bar{9}$ / $\ddot{1}$	SVM	0.498	0.181
DF	Req. Funct.	$\bar{9}$ / $\ddot{1}$	LR	0.325	0.272	$\bar{9}$ / $\ddot{8}$	SGD	0.33	0.174
	Clarification	$\bar{2}$ / $\ddot{1}$	LR	0.435	0.403	$\bar{2}$ / $\ddot{4}$	LR	0.435	0.298
	Solution	$\bar{9}$ / $\ddot{1}$	SGD	0.227	0.197	$\bar{9}$ / $\ddot{4}$	SGD	0.227	0.129
SE	Req. Funct.	$\bar{9}$ / $\ddot{1}$	LR	0.164	0.122	$\bar{9}$ / $\ddot{4}$	LR	0.164	0.078
	Clarification	$\bar{2}$ / $\ddot{1}$	LR	0.249	0.229	$\bar{2}$ / $\ddot{1}$	LR	0.249	0.161
	Solution	$\bar{9}$ / $\ddot{1}$	LR	0.117	0.099	$\bar{7}$ / $\ddot{6}$	MNB	0.125	0.066

Table IV summarizes average and best achieved F_1 scores (1) for each of the 7 ML algorithms, (2) for each of the 3 labels, and (3) for each of the 3 levels of detail. The results on the left hand side of Table IV are achieved using the BOW MLF only, whereas the right hand side shows the results for all linguistic MLF-sets. For cells marked with a gray background in Table IV, additional details with respect to the preprocessing settings can be found in Figure 2, which utilizes a combination scatter and box plots to show the influence of all preprocessing settings together with the related standard deviation, median and mean, on four representative examples.

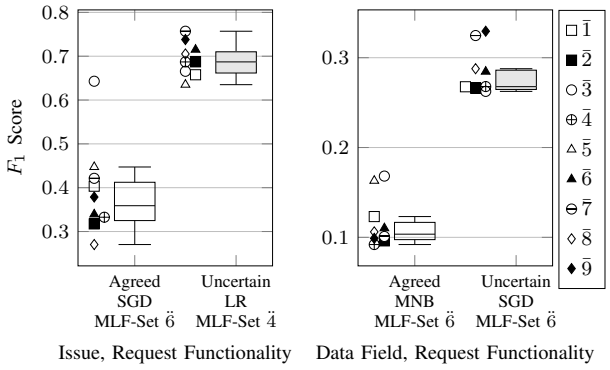

Fig. 2. Detailed Examples for Preprocessing Setting Influences

In comparison with other preprocessing settings, Setting $\bar{1}$

[9]Data sets, complementary figures, code, and a result database with SQL-queries for each RQ are available at: https://zenodo.org/record/56907.

[10]Preprocessing settings and MLF-sets need to be fixed for further experiments: calculating all permutations of preprocessing techniques, MLFs, levels of detail, and labels yields more than one year runtime on a standard laptop.

TABLE V
MACHINE LEARNING FEATURE SETS

MLF-Set	$\mathring{1}$	$\mathring{2}$	$\mathring{3}$	$\mathring{4}$	$\mathring{5}$	$\mathring{6}$	$\mathring{7}$	$\mathring{8}$	$\mathring{9}$	$\mathring{10}$	$\mathring{11}$	$\mathring{12}$	$\mathring{13}$	$\mathring{14}$	$\mathring{15}$	$\mathring{16}$	$\mathring{17}$	$\mathring{18}$
BOW	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
bi- & tri-grams	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓
SAO	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓	-	✓
data field	-	-	✓	✓	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
issue w/o type	-	-	-	-	✓	✓	-	-	-	-	✓	✓	-	-	✓	✓	-	-
issue with type	-	-	-	-	-	-	✓	✓	-	-	-	-	✓	✓	-	-	✓	✓
keywords	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	✓	✓	✓	✓

performed best in a single experiment run only. This indicates that lowering and stemming can be considered best practices in SFR detection, too. Preprocessing settings $\bar{7}$ and $\bar{9}$ deliver the best results in nine and ten cases respectively, covering 55% of the entire cases. Both, settings $\bar{7}$ an $\bar{9}$ employ all preprocessing techniques. Additionally, setting $\bar{7}$ separates technical data from NL. These results suggest that most preprocessing techniques have a positive influence on SFR detection. In 10 out of 36 cases, or 28%, settings $\bar{2}$ and $\bar{3}$, which include only lowering, stemming and stopword removal, achieve best F_1 scores. This implies that SFR detection can be approached without punctuation removal or the separation of technical data. However, there is no pattern in terms of detection level, label, or data set, and we cannot state that a specific preprocessing setting should always be employed: e.g. the detailed examples in Figure 2 show that the two preprocessing settings that perform best on the agreed upon data set, $\bar{2}$ [■] and $\bar{4}$ [⊕], perform bad on the uncertain cases and vice versa for the settings $\bar{7}$ [⊖] and $\bar{9}$ [⋄]. But even with this uncertainty, *preprocessing setting $\bar{9}$ will be employed to answer the remaining RQs* for two reasons: (1) it generally shows a reasonable performance on the random samples that were examined in detail, and (2) it results in the best reduction of MLF vectors, which saves memory, computation time, and accounts for better scalability.

In terms of ML models, Table IV shows that MNB performs best for the agreed upon cases, whereas LR performs better on the uncertain cases. There are two possible explanations for this: (1) LR is known, in essence, to outperform NB variants with more training data [27], and (2) more annotations are contained in uncertain data set and thus more linguistic MLFs are included in the training data. MNB assigns independent weights to repetitive MLFs that correlate with the label and with each other due to a conditional independence assumption. In contrast LR compensates such inter-MLF correlations, which may improve the prediction rate.

B. MLFs and Detection Levels (RQ2)

The previous Section included various linguistic MLF-sets to study appropriate preprocessing settings for SFR detection, shown in Table III (b). The right hand side in Table IV shows that settings $\ddot{6}$ (the combination of all linguistic MLFs), $\ddot{4}$ (BOW combined with bi- and tri-grams) and $\ddot{1}$ (BOW only) provide the best results across all preprocessing settings. In fact, only preprocessing settings that include the BOW MLF train usable models for SFR detection. E.g. on the agreed upon data set, BOW results in a detection rate between 34% and 49% on the issue level and 10% to 13% on the data field level. In contrast, MLF-sets without BOW do not exceed 10% on any level of detail . However, the combination of linguistic MLFs (e.g. MLF-sets $\ddot{4}$, $\ddot{5}$, and $\ddot{6}$) generally improves detection rates.

Table V summarizes the MLF-sets that are used for further experiments. We decided to include the linguistic MLF-sets $\ddot{1}$ and $\ddot{2}$ in all further MLF-combinations: MLF-set $\ddot{1}$ can be derived easily and quickly, having competitive performance, whereas MLF-set $\ddot{6}$ delivers best results. MLF-sets $\mathring{3}$ to $\mathring{10}$ extend the linguistic MLFs with data field meta-data, issue meta-data, and keywords. The issue type or tag might be a very strong indicator for a functionality request. Therefore, the issue meta-data is split up into two separate sets: (1) excluding the issue type or tags, and (2) including the issue type or tags. MLF-sets $\mathring{11}$ - $\mathring{18}$ are combinations of linguistic MLFs, data field meta-data, issue meta-data and keywords.

For each MLF-set, the box plots on the left hand side of Figure 3 visualize the F_1 scores over all ML models. White boxes represent the agreed upon cases, gray boxes the uncertain cases. In addition, a small symbol indicates the algorithms that performed best for the F_1 scores. The scatter plots on the right hand side of Figure 3 visualize the $\text{MAX}(R), P_{\geq p}$ scores for the ML models with the best score.

Similar to RQ1, the results for uncertain cases are better compared to the results of agreed upon cases. As already mentioned, our annotators did not annotate sentences they felt uncertain about so that these sentences did not make it in the agreed upon data set. However, we may claim that our annotators, in general, annotated SFRs correctly. Therefore, more presumably correct data might be found in the uncertain data set. Assuming that the algorithm detects most of these uncertain cases, this leads to fewer false positives and thus a higher precision, which is underpinned by the $\text{MAX}(R), P_{\geq 0.05}$ scores for the data field level in Figure 3.

The inclusion of keywords hardly impacts the results, clearly evident in the comparison between MLF-sets $\mathring{1}$ and $\mathring{2}$ with $\mathring{9}$ and $\mathring{10}$. The inclusion of data fields, on the other hand, and especially ITS meta-data do have a positive impact. However, additional combinations of data field and ITS meta-data do not necessarily improve the results, although a slight increase can be noticed on the data field level by comparing only the ITS meta-data MLFs ($\mathring{5} - \mathring{10}$) with combined MLFs ($\mathring{10} - \mathring{18}$).

The right hand side of Figure 3 shows that linguistic MLFs are generally sufficient whenever recall is to be maximized. On the issue level, the SVM model improves the precision by considering ITS meta-data (see $\mathring{5} - \mathring{8}$), while leaving the recall almost unaffected. The same slight improvement can be noticed in the F_1 scores on the left hand side for SGD and LR algorithms. Remarkably, using ITS features with or without issue type impacts detection results little, which renders the issue type, at least in the projects researched in this paper, a weaker predictor than initially assumed.

Overall, the combination of all MLF-types delivers the best performance in terms of F_1 score, whereas the linguistic MLFs

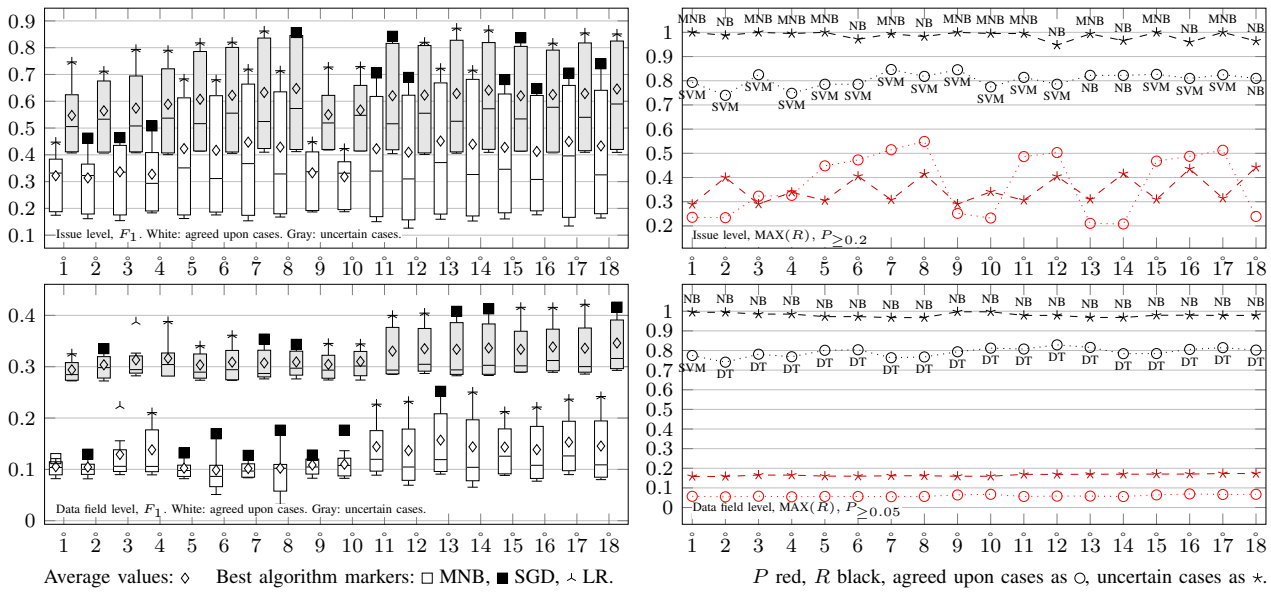

Fig. 3. *Request Functionality* Detection: Comparison of MLF-sets for Issue and Data Field levels with Different Classifiers.

on their own are competitive in terms of MAX(R), $P_{\geq p}$ scores.

The F_1 and MAX(R), $P_{\geq p}$ scores on issue level are higher compared to other levels. However, the data field level delivers competitive scores with respect to the MAX(R), $P_{\geq p}$ measure, especially on the uncertain data set. This is important for the practical application of the approach: detecting SFRs on the data field level implies less manual work than a detection on the issue level. On the sentence level (omitted in Figure 3), the worst results are generated. For example for MAX(R), $P_{\geq 0.05}$ the recall was only slightly over 60% in the agreed upon cases. For the uncertain cases, however, a recall of 100% can be achieved for MAX(R), $P_{\geq 0.05}$ even on the sentence level with F_1 scores of about 11%.

The measurements in this section are exemplary for the other two labels. *Clarification*s are detected with similar P and R. *Solution*s are detected with inferior detection rates compared to the other labels, but the results show the same characteristics with respect to the MLF-sets[11]. Table VI summarizes the best achieved F_1 and MAX(R), $P_{\geq p}$ scores with the related ML models and MLF-sets for all the labels on every level of detail.

Finally, combining MLF-sets generally increases the standard deviation for the F_1 scores across all models as shown on the left hand side of Figure 3. Hence, the algorithm choice (or an evaluation of multiple algorithms) is important whenever features are combined or results may deteriorate.

C. Cross-training (RQ3)

To answer this RQ, three of the four project data sets were used for training, the fourth project for detection. Table VII shows the best achieved F_1 and MAX(R), $P_{\geq p}$ scores with according MLF-sets for the *request functionality* label. Again, similar to RQ2, *request functionality* is representative for the other labels.

[11]Graphs for all levels of detail and all labels are included in the download.

TABLE VI
BEST F_1 AND MAX(R), $P_{\geq p}$ SCORES FOR ALL LEVELS AND LABELS

level	label	MLF set	alg.	values	MLF set	alg.	values
		agreed		F_1	**uncertain**		F_1
I	RF	18	SGD	0.74	13	LR	0.87
I	C	14	SVM	0.74	9	LR	0.78
I	S	15	SGD	0.54	7	SVM	0.75
DF	RF	13	SGD	0.25	17	LR	0.42
DF	C	4	MNB	0.15	17	SVM	0.45
DF	S	9	MNB	0.09	17	SGD	0.27
Se	RF	13	LR	0.13	18	SGD	0.19
Se	C	4	MNB	0.08	4	MNB	0.25
Se	S	1	MNB	0.05	4	MNB	0.13
MAX(R), $P_{\geq p}$		**agreed**		p R P	**uncertain**		p R P
I	C	7	SVM	0.2 0.84 0.66	7	NB	0.2 1.0 0.28
I	RF	7	SVM	0.2 0.85 0.51	1	MNB	0.2 1.0 0.29
I	S	17	SVM	0.2 0.74 0.39	7	SVM	0.2 0.78 0.75
DF	C	14	NB	0.05 0.99 0.05	3	NB	0.05 0.99 0.25
DF	RF	12	DT	0.05 0.83 0.06	9	NB	0.05 0.99 0.16
DF	S	1	MNB	0.05 0.29 0.07	17	NB	0.05 0.98 0.11
Se	C	4	MNB	0.05 0.18 0.05	3	NB	0.05 0.99 0.13
Se	RF	3	MNB	0.05 0.62 0.06	3	NB	0.05 1.0 0.06
Se	S	2	MNB	0.05 0.03 0.1	1	MNB	0.05 0.93 0.06

with: RF=*Request Functionality*, C=*Clarification*, S=*Solution*

Comparing these results to the results with 10-fold-cross evaluation from Table VII, cross-training delivers very similar F_1 and MAX(R), $P_{\geq p}$ scores with a variability of $\pm 5\%$, even though a lower amount of training data is available for cross-training than for 10-fold-cross validation.

VII. Discussion

This section discusses the main implications of our study for future research on SFR detection in ITSs:

TABLE VII
BEST CROSS-TRAINING F_1 AND MAX$(R), P_{\geq p}$ SCORES FOR REQUEST FUNCTIONALITY

project	level	MLF-set	alg.	values			MLF-set	alg.	values		
			agreed	F_1		avg.		**uncertain**	F_1		avg.
cg	I	$\mathring{1}\mathring{1}$	SGD	0.74			$\mathring{1}\mathring{5}$	SGD	0.85		
li	I	$\mathring{7}$	LR	0.76		0.72	$\mathring{6}$	LR	0.84		0.87
ra	I	$\mathring{5}$	LR	0.67			$\mathring{1}\mathring{1}$	SGD	0.86		
re	I	$\mathring{1}\mathring{4}$	SVM	0.70			$\mathring{1}\mathring{7}$	SGD	0.91		
cg	DF	$\mathring{1}\mathring{3}$	SGD	0.29			$\mathring{1}\mathring{7}$	SGD	0.26		
li	DF	$\mathring{1}\mathring{8}$	LR	0.42		0.31	$\mathring{1}\mathring{8}$	LR	0.41		0.39
ra	DF	$\mathring{1}\mathring{4}$	SGD	0.24			$\mathring{1}\mathring{5}$	SGD	0.36		
re	DF	$\mathring{1}\mathring{4}$	SGD	0.29			$\mathring{1}\mathring{4}$	SGD	0.52		
cg	Se	$\mathring{4}$	MNB	0.17			$\mathring{1}\mathring{1}$	SGD	0.13		
li	Se	$\mathring{1}\mathring{8}$	SGD	0.27		0.19	$\mathring{1}\mathring{7}$	LR	0.16		0.19
ra	Se	$\mathring{1}\mathring{3}$	LR	0.16			$\mathring{1}\mathring{6}$	SGD	0.18		
re	Se	$\mathring{3}$	MNB	0.16			$\mathring{4}$	MNB	0.27		
MAX$(R), P_{\geq p}$			**agreed**	p	R	P		**uncertain**	p	R	P
cg	I	$\mathring{3}$	NB	0.2	0.89	0.22	$\mathring{1}$	MNB	0.2	1.00	0.24
li	I	$\mathring{3}$	SGD	0.2	1.00	0.38	$\mathring{7}$	NB	0.2	1.00	0.21
ra	I	$\mathring{1}$	SVM	0.2	0.80	0.22	$\mathring{1}$	NB	0.2	1.00	0.33
re	I	$\mathring{1}\mathring{3}$	NB	0.2	0.97	0.27	$\mathring{1}\mathring{3}$	NB	0.2	1.00	0.59
cg	DF	$\mathring{1}$	RF	0.05	0.71	0.06	$\mathring{9}$	NB	0.05	0.99	0.13
li	DF	$\mathring{4}$	DT	0.05	1.00	0.06	$\mathring{7}$	NB	0.05	1.00	0.08
ra	DF	$\mathring{1}\mathring{3}$	SGD	0.05	0.53	0.14	$\mathring{1}$	NB	0.05	1.00	0.10
re	DF	$\mathring{9}$	NB	0.05	1.00	0.06	$\mathring{7}$	NB	0.05	1.00	0.19
cg	Se	$\mathring{1}$	MNB	0.05	0.52	0.05	$\mathring{5}$	NB	0.05	1.00	0.05
li	Se	$\mathring{1}\mathring{3}$	SGD	0.05	0.89	0.07	$\mathring{1}\mathring{6}$	DT	0.05	0.93	0.05
ra	Se	$\mathring{1}\mathring{2}$	SGD	0.05	0.40	0.06	$\mathring{1}$	MNB	0.05	0.98	0.06
re	Se	$\mathring{3}$	SGD	0.05	0.71	0.06	$\mathring{7}$	NB	0.05	1.00	0.07

with: cg=c:geo, li=Lighttpd, ra=Radiant, re=Redmine

Preprocessing techniques should be employed. Section VI-A shows that the application of preprocessing techniques improves F_1 as well as MAX$(R), P_{\geq p}$ scores in context of SFR detection. Even stop-word removal improves the results, although coders considered stop-words such as *should*, *could*, or *would* relevant for classification.

Enough annotations should be available for training and evaluation. In Section VI we report on models trained on $36 \times \frac{9}{10}$, $76 \times \frac{9}{10}$ and $100 \times \frac{9}{10}$ annotations for the agreed upon data set. . The F_1 and MAX$(R), P_{\geq p}$ scores increase relative to the amount of training data.

Simple linguistic features are sufficient. Sets that add bi-, tri-grams and SAO to BOW (i.e. sets with even numbers in Figure 3) show almost no improvement. If computation time or the amount of features need to be reduced, complexity can be downsized at this point.

MLFs derived from meta-data improve detection rates. Sets $\mathring{1}\mathring{1}$-$\mathring{1}\mathring{8}$ in Figure 3 show that the inclusion of MLFs derived from meta-data improve detection rates. Further research (e.g. ML feature selection techniques [7]) is needed to identify the exact MLFs with the highest impact.

The combination of ML algorithms and MLF-sets is important. Different ML algorithms work differently. For example, NB treats correlating MLFs independently, LR compensates such correlations [18]. The lower whiskers in Figure 3 indicate that some ML algorithms do not profit from additional features. Hence, the combination of ML algorithms and MLF-sets is important to consider.

ML algorithms should be selected according to the data mining goal. Figure 3 shows that different ML algorithms are responsible for the best F_1 and the best MAX$(R), P_{\geq p}$ sores. Consequently, algorithms performing best in order to maximize the SFR detection rate might not perform best in order to balance detection rate and precision.

Trained models can be re-used for other projects. Section VI-C reveals that cross-training works competitive to 10-fold-cross validation for *request functionality* and *clarification* labels. However, this does not not hold true for every project and thus needs further evaluation or replication by additional studies.

In some projects SFRs are likely composed of NL patterns. In five cases setting $\mathring{9}$ (BOW and manually compiled keyword lists) achieved the best F_1 or MAX$(R), P_{\geq p}$ scores, as shown in Tables VI and VII. This indicates that at least some *request functionalities* are described with reoccurring words or certain NL patterns. As for other requirements [16], such patterns would be a powerful heuristic and/or ML feature for SFR detection.

Using only the agreed upon cases from dual coded data can be inferior for ML. In almost each case, better predictions can be made for the uncertain cases. This eases the practical applicability of the approach, as multiple coding is not applicable in industry, or generally in practice, keeping the required effort in mind.

On experiment runtime: The runtime on a standard laptop is reasonable, considering that the code was only slightly optimized for speed. For example, an experiment using BOW features, 3 classifiers, and 599 issues takes < 10 seconds. An experimental run including all MLFs takes < 10 minutes. . These measures exclude the calculation of typed dependencies, which takes about an additional hour and is needed only for the SAO feature[12].

The limitations of these implications are discussed in the next section.

VIII. LIMITATIONS AND THREATS TO VALIDITY

Considering the internal validity, manual data annotation involves the risk that human coders do make mistakes. This can result in unusable ML models and thus influence predictions and results. To mitigate this risk we deployed best practices in content analysis [28]. In particular, we established a coding guidebook and discussed annotations on test data before the actual coding. Although we coded on the level of sentences, we achieved reasonable kappa scores. Finally, we created one data set with only those annotations on which two coders agreed. However, the coders had a lower agreement on comments. This could be due to increasing tiredness or even inadvertence. Annotating up to 50 comments in a single issue is an arduous task. A low agreement might lead to missing training and validation data and thus to decreasing detection rates and/or increasing false positive rates.

[12]Typed dependencies are cached, yielding a $>$ factor 100 speedup.

However, the assessment whether a sentence describes a request for software functionality or not is arguably subjective. Although the agreed upon data set has a higher scientific quality, using the data created by a single annotator is not necessarily bad. E.g. the perception of a single analyst might be exactly what an ML model should replicate when applied in an industry project. We addressed this subject by including the uncertain cases in our experiments. Finally, the overall size of our gold standard is limited. Although we analyzed over $10,000$ sentences in over 4500 ITS data fields, our gold standard contains only 76 *request functionality*, 36 *solution* and 100 *clarification* labels for the agreed upon cases. Although we received promising results on the issue and data field level, other research indicates that more than 150 labeled instances are necessary for a reliable classification [20]. Hence, this study should be seen as a first step in SFR detection and needs further replication or extension studies.

Considering the external validity, we cannot ensure that our results can be transferred to ITS data of other projects[13]. It is still likely that the overall approach is transferable, since we intentionally employed projects with a broad range of target groups, programming languages, etc. In a more consistent data set even better results can be expected.

IX. Conclusion and Future Work

In this paper multiple preprocessing techniques, machine learning algorithms, and machine learning features are evaluated to detect software feature requests in issue tracking systems. By introducing the $\text{MAX}(R), P_{\geq p}$ measure, we find that some algorithms maximize the F_1 score whereas others maximize recall. Furthermore, we show that software feature requests detection can be approached on the level of issues and data fields with satisfactory results, but the exact sentences that describe the software feature requests are hard to find. Further research on this topic can tackle multiple challenges: machine learning feature sets can be curtailed further to determine which exact machine learning feature has the highest impact on software feature request detection. The experiment can be replicated or extended with additional data or machine learning features. Finally, multiple machine learning models or the detection on different levels can be combined. To ease such efforts, we distribute all our data sets, code and results along with this paper.

References

[1] A. Nguyen Duc, D. S. Cruzes, C. Ayala, and R. Conradi, "Impact of Stakeholder Type and Collaboration on Issue Resolution Time in OSS Projects," in *OSS Syst.: Grounding Research*, 2011, vol. 365, pp. 1–16.

[2] B. Paech, P. Hübner, and T. Merten, "What are the Features of this Software?" in *ICSEA 2014, The Ninth Int. Conf. on Software Eng. Advances*. IARIA XPS Press, 2014, pp. 97–106.

[3] T. Merten, B. Mager, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech, "Requirements Communication in Issue Tracking Syst. in Four Open-Source Projects," in *6th Int. Workshop on Requirements Prioritization and Communication (RePriCo)*. CEUR Workshop Proc., 2015, pp. 114–125.

[13] In an industrial ITS we encountered the "coffee break issues". They include information similar to "As discussed during coffee break". Obviously, this is insufficient for text analysis.

[4] K. Herzig, S. Just, and A. Zeller, "It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction," in *Proc. of the 2013 Int. Conf. on Software Eng. (ISCE)*. IEEE Press, 2013, pp. 392–401.

[5] I. Witten, E. Frank, and M. A. Hall, *Data Mining*. M. Kauffmann, 2000.

[6] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[7] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The J. of Mach. Learning Research*, vol. 3, pp. 1157–1182, 2003.

[8] P. Domingos, "A few useful things to know about machine learning," *Commun. of the ACM*, vol. 55, no. 10, p. 78, 2012.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Mach. Learning in Python," *J. of Mach. Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] S. Bird, E. Klein, and E. Loper, *Natural Language Proc. with Python*, 1st ed. O'Reilly Media, Inc., 2009.

[11] I. Skerrett and The Eclipse Foundation, "The Eclipse Community Survey 2011," 2011.

[12] D. Berry, R. Gacitua, P. Sawyer, and S. F. Tjong, "The Case for Dumb Requirements Eng. Tools," in *18th Intl. Working Conf. on Req. Eng.: Foundation for Software Quality (REFSQ 2012)*, vol. 7195 LNCS. Essen, Germany: Springer, 2012, pp. 211–217.

[13] N. H. Bakar, Z. M. Kasirun, and N. Salleh, "Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review," *J. of Syst. and Software*, vol. 106, pp. 132–149, aug 2015.

[14] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests," in *Proc. of the 2008 conference of the center for advanced studies on collaborative research meeting of minds - CASCON '08*. ACM Press, 2008, p. 304.

[15] I. Chawla and S. K. Singh, "An Automated approach for Bug Categorization using Fuzzy Logic," in *Proc. of the 8th India Software Eng. Conf. - ISEC '15*. ACM Press, 2015, pp. 90–99.

[16] R. E. Vlas and W. N. Robinson, "Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open-Source Software Development Projects," *J. of Management Information Syst.*, pp. 1–39, 2012.

[17] E. Guzman, "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews," *mobis.informatik.uni-hamburg.de*, pp. 153–162, 2014.

[18] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Proc.*. MIT Press. Cambridge, MA, 1999.

[19] C. Fitzgerald, E. Letier, and A. Finkelstein, "Early failure prediction in feature request management systems: an extended study," *Requirements Eng.*, vol. 17, no. 2, pp. 117–132, apr 2012.

[20] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *2015 IEEE 23rd Int. Requirements Eng. Conf. (RE)*. IEEE, aug 2015, pp. 116–125.

[21] A. Fantechi and E. Spinicci, "A Content Analysis Technique for Inconsistency Detection in Software Requirements Documents," in *Proc. of the VIII Workshop on Requirements Eng. (WER)*, 2005, pp. 245–256.

[22] D. Chen and C. D. Manning, "A Fast and Accurate Dependency Parser using Neural Networks," in *Proc. of the Conf. on Empirical Methods in Natural Language Proc. (EMNLP)*, no. i, 2014, pp. 740–750.

[23] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educ. and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, apr 1960.

[24] M. Porter, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 14, no. 3, pp. 130–137, 1980.

[25] T. Merten, B. Mager, S. Bürsner, and B. Paech, "Classifying unstructured data into natural language text and technical information," in *Proc. of the 11th Work. Conf. on Mining Software Repositories - MSR 2014*. ACM Press, 2014, pp. 300–303.

[26] R. Feldman and J. Sanger, *The Text Mining Handbook*. Cambridge: Cambridge University Press, 2006.

[27] A. Y. Ng and M. I. Jordan, "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Inf. Proc. Syst. 14 (NIPS 2001)*, 2001, pp. 841–848.

[28] K. A. Neuendorf, *The Content Analysis Guidebook*. SAGE, 2002.