Locating Requests among Open Source Software Communication Messages

Ioannis Korkontzelos, Sophia Ananiadou
National Centre for Text Mining
School of Computer Science
The University of Manchester
{Ioannis.Korkontzelos, Sophia.Ananiadou} @manchester.ac.uk

Abstract

As a first step towards assessing the quality of support offered online for *Open Source Software (OSS)*, we address the task of locating *requests*, i.e., messages that raise an issue to be addressed by the OSS community, as opposed to any other message. We present a corpus of online communication messages randomly sampled from newsgroups and bug trackers, manually annotated as requests or non-requests. We identify several linguistically shallow, content-based heuristics that correlate with the classification and investigate the extent to which they can serve as independent classification criteria. Then, we train machine-learning classifiers on these heuristics. We experiment with a wide range of settings, such as different learners, excluding some heuristics and adding unigram features of various parts-of-speech and frequency. We conclude that some heuristics can perform well, while their accuracy can be improved further using machine learning, at the cost of obtaining manual annotations.

Keywords: online dicussion threads, online message classification, corpus of online messages

1. Introduction

Assessing the quality of *Open Source Software (OSS)* is useful for deciding whether a project meets certain standards for adoption. It is a complex task, since a multitude of evaluation aspects, such as maturity, activity of development and user support need to be considered (Spinellis et al., 2009; Bianco et al., 2010; Samoladas et al., 2008; Fuggetta, 2003). The task becomes even more challenging when one needs to choose among many OSS projects that offer similar functionality, based on objective evidence (Dahlander and Magnusson, 2005). For example, there are more than 20 open source XML parsers for the Java programming language[1].

Analysing source code repositories can provide information about evaluation aspects, such as software maturity, activity of development and quality of comments. Exploring relevant communication channels, such as newsgroups and bug tracking systems, is useful for assessing the level of user support, interest and satisfaction for an OSS project (Scacchi, 2007). For example, we can compute metadata such as the number of experts and active users, the number of open bugs, the rate of fixing bugs and whether user questions are answered timely and satisfactorily (Lakhani and von Hippel, 2003).

Computing such metadata requires knowing firstly which messages pose questions or requests as opposed to answers, comments, announcements and other messages. Requests are messages that raise an issue to the OSS community, asking for help, reporting some installation problem or other bug or proposing an improvement. We present a corpus of $1,030$ messages relevant to OSS, randomly selected from a variety of channels, and also our experiments to classify request vs. non-request messages. We investigate the extent to which simple, linguistically shallow, content-based, observational heuristics can serve as unsupervised classifiers to locate requests. Then, we use these heuristics to train supervised learners. Results show that some heuristics achieve accuracies that exceed significantly the major class baseline. Machine learning can improve this accuracy at the cost of obtaining manual annotations. In the future, we plan to investigate features based on information other than message content, such as the position of a message in a thread. We also plan to investigate how non-request messages can be further classified in more fine-grained categories.

The remainder of this paper is structured as follows: In section 2., we summarise related literature and in section 3., we present the corpus of OSS-related online communication messages, with details about the annotation of requests and non-requests and some examples indicating that the binary classification task is non-trivial. Section 4. describes heuristic methods that correlate with the request vs. non-request classification and can be used as unsupervised classifiers or as features for trainable classifiers. Section 5., presents evaluation settings and results, and we conclude in section 6..

2. Related literature

The most relevant work in the literature is Wang et al. (2010), which performed thread-level analysis of CNET technical forums. They performed classification experiments based on a corpus and a classification scheme. In contrast, we exploit OSS-related forums specifically and we investigate only content-based features. Wang et al. (2011) and Wang and Rosé (2010) used an extended set of features to predict discourse relations in forum threads.

Other relevant work includes Ding et al. (2008) that used conditional random fields to match answers and content messages to questions in online forums. Baldwin et al. (2007) and Baldwin et al. (2010) classified entire Linux forum threads in classes related to task specificity, completeness and solvedness. Messages were classified as problems, solutions or miscellaneous and then classified into further

[1] Open source XML parsers in Java:
java-source.net/open-source/xml-parsers

System	Channels	Messages	Requests	Non-requests
NNTP	4	208	77	131
Bugzilla	8	410	131	279
Github	22	412	98	314
Total	34	1030	306	724

Table 1: Online communication corpus statistics. Channels correspond to NNTP newsgroups, Bugzilla products or GitHub projects.

subclasses.

Ding et al. (2008) proposed a general framework based on Conditional Random Fields to detect context messages and answers of questions in forum threads. Kim et al. (2010) attempted to classify forum posts according to dialogue acts and, then, structured thread messages as a discourse. Hassan et al. (2010) proposed a method for detecting attitudinal sentences in threads, as well as types of attitude. Lui and Baldwin (2009) drew the profile of users based on the properties of their activity in online forums.

Krishnamurthy (2005) presented empirical evidence from 100 mature *OSS* projects and drew associations between the number of software developers per project, its age and its popularity. Crowston and Howison (2005) and Bagozzi and Dholakia (2006) presented similar studies focussing on social and communication structures concerning *OSS*.

3. Corpus of online communication messages

$1,030$ online communication messages[2] were randomly selected from Network News Transfer Protocol (NNTP) newsgroups, Bugzilla and Github. NNTP is a protocol for transporting, posting and sending articles between servers and client applications. Bugzilla is a bug tracking system, while GitHub is a web-based hosting service for software development, which hosts its own bug tracking system, GitHub Issue Tracker. NNTP articles, Bugzilla and GitHub comments were selected randomly so that the sample exhibits similar characteristics to the population as a whole.

Table 1 presents high level statistics of the corpus[3], while table 2 shows the NNTP newsgroups, Bugzilla and GitHub projects that were selected. The corpus is stored in XML format. For each message, apart from the textual content, additional identity information and metadata, such as author and date, are stored.

3.1. Corpus annotation

Each message was annotated manually as request or non-request by a computational linguist. The fundamental criterion for annotating a message as request is whether it raises a new topic to the OSS community. Namely, a request might:

- report a newly discovered bug.
- ask whether a previously reported bug has been fixed.
- report a difficulty in installing or using the OSS project.
- state that the user is facing a previously reported problem, bug or difficulty.
- propose an improvement for the OSS project.

All other messages are considered as non-requests. As shown in the two rightmost columns of table 1 there are more non-requests than requests. The main reason is that the corresponding content range is wider, since it contains:

- All communication between developers and users after a bug is reported and before it is fixed.
- User non-requests after the bug is fixed or it is decided that it cannot be fixed.
- Communication between developers about their progress on fixing bugs and improving the OSS.
- Communication between developers for assigning bugs and jobs to each other.
- Notices and announcements about news, new releases or other changes concerning an OSS.

These definitions are driven by the usability of this classification towards evaluating OSS projects. Since the overall target is to score online communications in terms of quality of provided user support, we choose to classify to-

request 1	**server URL**	news.eclipse.org
	newsgroup	eclipse.hudson
	text	I just started the Tomcat server, which is hosting Hudson, using jdk1.6.0_27 as JAVA_HOME. I still get the error, when Hudson attempts to send mails.
request 2	**server URL**	news.eclipse.org
	newsgroup	eclipse.hudson
	text	I do have the groovy-support plugin installed. I reduced the groovy script to only a single println and even that failed.
request 3	**server URL**	bugzilla.redhat.com
	project	Red Hat Linux
	text	tin in its default configuration tries to read usr/lib/news/active, when a newsspool operation is requested. This is incorrect, active is in /var/lib/news under RH.
non-req. 1	**server URL**	news.eclipse.org
	project	eclipse.hudson
	text	What version of jboss as are you deploying to (According to the stack trace it's version 5x)?

Table 3: 3 examples of a requests, where no question mark or *WH* question word is used, and an example of a non-request that includes a question mark and a *WH* question word. The latter is a question asked by a developer to the user that has previously submitted a request.

[2]The corpus is freely available, under the Apache License. Please, contact the authors for details.

[3]Fewer newsgroup articles than Bugzilla and Github comments were selected, because the corresponding projects were among the interests of the industrial partners involved in OSSMETER (STREP EU project, grant number 318736).

NNTP server	NNTP newsgroups (# articles)
news.eclipse.org	eclipse.technology.subversive (52), eclipse.hudson (52), eclipse.platform (52)
news.gmane.org	gmane.comp.java.sonar.general (52)
Bugzilla server	**Bugzilla projects (# articles)**
bugzilla.redhat.com	gmane.comp.java.sonar.general (52), Bugzilla (52), Fedora (52), Issue-Tracker (52) Pulp (52), Red Hat Database (46), Red Hat Enterprise Linux 7 (52) Red Hat Linux (52), Topic Tool (52)
GitHub server	**GitHub projects (# comments)**
api.github.com/ repositories	acts_as_geocodable (5), amazon-ec2 (6), attachment_fu (43), audited (15), braid (6) capsize (4), cache_fu (3), chronic (56), enum_field (1), eycap (11), forgery (16), git-wiki (4) god (23), grit (24), low-pro-for-jquery (1), resource_controller (8), restful-authentication (23) rubinius (56), ruby-git (23), ruby-on-rails-tmbundle (25), signal-wiki (3), thin (56)

Table 2: NNTP newsgroups, Bugzilla and GitHub projects included in the online communication corpus. The number of NNTP articles, Bugzilla and GitHub comments appears within parentheses.

gether messages that contribute similarly to the evaluation score, positively or negatively. In particular, request messages are considered to negatively affect the quality score: many requests paired with few non-requests probably indicate that the developers cannot adequately support users. In contrast, many non-requests indicate increased activity levels and probably the successful handling of user requests. However, very few or zero requests may indicate absence of user interest, and thus, have a negative impact on the quality score. The effect of the number of request and non-request messages will feed into a thread classification system. Then, we will be able to measure the number of unaddressed requests and the actual time between a request and the corresponding non-request.

3.2. Classification challenges

Classifying messages automatically is challenging for a number of reasons. Question marks and/or *WH* question words, i.e. words that introduce questions, such as *"what"*, *"who"* and *"where"*, can indicate requests. Question marks are only present in direct interrogative sentences, however, in written form, questions are typically expressed indirectly. In indirect questions, *WH* question words are present.

The corpus contains noteworthy exceptions to the hypothesis that request messages contain question marks or *WH* question words. Table 3 presents 3 request messages, none of which contains direct or indirect interrogative sentences. In request 1, it is mentioned that the user encountered some error. Similarly, request 2 and 3 express deficiencies by the words *"failed"* and *"incorrect"*. In contrast, table 3 presents a non-request that contains both a question mark and a WH question word. It is a direct interrogative sentence that concerns communication between the developer that attempts to address a request and the user that reported the deficiency.

Table 4 shows an example non-request and 2 example requests. Non-request 2 expresses a question submitted by a developer that attempts to address a request submitted previously. The actual request message is included in the message body, indented with the greater-than symbol (>). This feature is typical in email replies and also adopted many bug tracking systems and newsgroups. To prevent text of previous messages from fooling a classifier, a cleaning step can be applied to exclude these lines of text, before presenting message content to the classifier.

Requests 4 and 5 in table 4 are difficult to recognise automatically. Request 4 contains no question mark or WH question words. It consists of a copied and pasted description of an error and a note of the sender that describes how the problem should be addressed, in their opinion. Request 5 consists of an installation log in which an error has occurred. The last line of the message is manually typed and mentions pieces of software already installed. The user intends to point out a contradiction: the installation fails although prerequisite software is already installed.

4. Heuristic classification methods

In this section, we present heuristic methods that correlate significantly with the classification scheme, inspired by observing instances, such as the ones presented in section 3.. Then, we discuss the feature set based on these heuristic methods, that has been used to train machine learners.

We investigate how well simple characteristics of online communication messages can serve as classification criteria. Since they require no linguistic preprocessing, the resulting classification methods are fast and suitable for online processing. The basic classification methods are:

Question mark method Classifies as requests messages that contain question marks, or as non-requests otherwise.

RE method Classifies as non-requests messages whose subjects start with *"RE: "* or *"Re: "*, or as requests otherwise. This method can be applied to NNTP newsgroup articles, only, since Bugzilla and GitHub comments have no subject.

Question words method A generalisation of the question mark method, so as to capture indirect questions in addition to direct ones. Messages that contain the WH question words *what*, *when*, *where*, *which*, *who*, *whom*, *whose*, *why* and *how* are classified as requests, otherwise as non-requests. Observing the corpus, we also added the words *help* and *please*, that are typically present in requests. Matching is performed in a case insensitive manner.

These three methods can be combined with a preprocessing step to *clean* text indented with the greater than symbol

non-request 2	**server**	api.github.com/repositories
	project	rubinius
	text	> is that needed at all for this change? No, this is feature from ruby 2.0. Each element in '$LOAD_PATH ' is frozen and '$LOAD_PATH' is cached in exactly the same way. So I think we can remove that part from pull request. > Also, is there a reason for all the synchronization in it? I'm not sure - I was based on the implementation of '$LOAD_FEATURES'
 > I think we should address that separately so we don't mix different changes, > which makes discussing and reviewing them harder. Agree, I will update this pull request and I will remove all changes related to '$LOAD_PATH'
request 4	**server**	bugzilla.redhat.com
	project	Topic Tool
	text	Description of problem: Tool gracefully ignores the fact that a topic could not be read (sometimes due to a validity issue) and keeps on retrieving subsequent topics: [Fatal Error] :234:6: The element type "step" must be terminated by the matching end-tag "</step >". ERROR: Unable to load topic (http://topicrepo.englab.bne.redhat.com/TopicRepository/Tasks/IPA/Installing_the_IPA_Server.xml). ERROR: Unable to parse 'Installation_Guide_Export/en-US/Infrastructure.xml'. Ideally I think we want to allow this but keep track of which topics couldn't be read and display a list at the end of the run to highlight that it wasn't successful.
request 5	**server**	bugzilla.redhat.com
	project	Red Hat Linux
	text	checking host system type... i586-pc-linux checking target system type... i586-pc-linux checking for gcc... gcc checking whether the C compiler (gcc) works... no configure: error: installation or configuration problem: C compiler cannot create executables. I have egcs and gcc installed.

Table 4: Examples of a non-request and 2 requests. Non-request 2 contains the request text submitted previously indented with the *greater-than symbol (>)*. Request 4 contains a copied and pasted description of an error. Request 5 consists of just an installation log in which an error has occurred.

(>). As discussed in section 3.2., this symbol is indicative of previous messages included in the current message. **Cleaning** is a method component rather than a method itself.

In addition, the basic classification methods can be combined with each other:

Question mark or words method Classifies as requests messages that contain question marks or WH question words, or as non-requests otherwise.

RE Question mark method Classifies messages as requests if their subjects do not start with *"RE: "* or *"Re: "*. The remaining messages are classified as requests if they contain a question mark, or as non-requests otherwise. This method applies to NNTP newsgroup articles, only.

RE Question mark or words method A combination of the two methods above. NNTP newsgroup articles that are not classified as requests by the *RE method* are classified as requests if they contain *question marks or WH question words*. Otherwise, articles are classified as non-requests.

All methods are developed for messages in English. However, they can be applied to other languages directly or after some minor modifications. In particular, **Cleaning** can be applied directly to any language, because greater-than (>) indentation is language independent. The remaining methods can be applied to other languages after translation. The **Question mark method** should be modified to capture the symbols that encode direct questions. The **RE method** should be modified to capture initials or prefixes that denote non-requests and are used in the subject of non-request emails. The **Question words method** should be modified to capture words that introduce questions in other languages.

Apart from applying the above heuristics as independent classification methods, we also use them as features for trainable machine learning classifiers. The feature set is of heuristic-based features is enriched by unigram features, hypothesising that occurrence of particular words might be indicative of request or non-request messages.

5. Evaluation

In this section, we present our experimentation with simple heuristics, disucssed in section 4., as independent unsupervised classification criteria. Then, we combine these criteria with unigram features to train machine learners. The motivation for our experiments is to inspect whether simple heuristics can successfully address the task and whether it is worth training machine learners to investigate correlations between the criteria and/or unigram features. In section 5.1. we discuss evaluation details, in section 5.2. we present evaluation results and in section 5.3. we discuss our experimentation conclusions.

#	Method				Classification accuracy (%)			
	Cleaning	Bugzilla	Github	Newsgroups	Bugzilla	Github	Newsgroups	Corpus
1	✗	qm			67.56	75.00	51.92	67.38
2	✗	qw			**69.76**	63.84	44.71	62.33
3	✗	qm qw			67.81	63.59	47.12	61.94
4	✗	qm	qm	re	67.56	75.00	**81.73**	**73.40**
5	✗	qm	qm	re qm	67.56	75.00	55.77	68.16
6	✗	qm qw	qm qw	re qm qw	67.81	63.59	47.12	61.94
7	✓	qm			67.56	75.49	**64.90**	70.19
8	✓	qw			**70.49**	64.08	52.40	64.27
9	✓	qm qw			**68.54**	64.08	54.81	63.98
10	✓	qm	qm	re	67.56	75.49	**81.73**	**73.59**
11	✓	qm	qm	re qm	67.56	75.49	**68.27**	**70.87**
12	✓	qm qw	qm qw	re qm qw	**68.54**	64.08	54.33	63.88
-	*Baseline:* Major class				68.05	76.21	62.98	70.29

Table 5: Evaluation results of heuristic methods used as classification criteria. *qm*, *qw* and *re* stand for the *question mark method*, the *question words method* and the *RE method*, respectively. Concatenations of these symbols denote combined methods. The last row reports the major class baseline. Scores that exceed the baseline are in bold face.

5.1. Experimental settings

We evaluated all heuristic methods with or without *Cleaning* as a preprocessing step. Since the *RE method* is applicable to newsgroup articles only, we paired it with others applicable to Bugzilla and Github comments, in order to allow the computation of results over the entire corpus.

For machine learning experiments, we employed the *Support Vector Machine (SVM)* (Cortes and Vapnik, 1995) and *Random Forest* (Breiman, 2001) implementations of the *WEKA toolkit* (Hall et al., 2009) and considered 10-fold cross validation. We employed the *linear* and *radial basis function (RBF)* SVM kernels, and used the default parameter *Gamma* value, 0.01, for the latter kernel. In addition to all 6 heuristics as features, we experimented with all combinations of omitting one or more during training. The number of experiments is equal to the number of all possible combinations for our 6 methods plus one experiment for no omissions:

$$\sum_{i=1}^{5} \left(\begin{array}{c} 6 \\ i \end{array} \right) + 1 = 63 \quad (1)$$

The feature space was enriched with unigrams that do not occur in a typical English stoplist and occur more frequently than a threshold *T*. We evaluated a range of values for *T*: $[1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$. Apart from all parts-of-speech (AllPoS), we experimented with unigram features of some parts-of-speech, only, i.e. just nouns (N), nouns and adjectives (NJ), nouns, adjectives and verbs (NJV). Moreover, we considered three feature value types for unigram features: binary, frequency and tf-idf (Salton and Buckley, 1988).

Due to the binary classification nature of these experiments, accuracy, i.e. the number of correctly predicted instances over the total number of instances is adequate to measure performance. To compute accuracy we used the Java class PrecisionRecallEvaluation, part of the *LingPipe API*[4].

[4]LingPipe 4.1.0 URL: alias-i.com/lingpipe

5.2. Experimental results

Table 5 shows accuracies achieved by 12 methods evaluated on the corpus of online communication messages, introduced in section 3.. The *RE method* and combined methods that use it are only evaluated on newsgroup articles, because the subject field is not available for Bugzilla and Github messages. To measure the effect of these methods on the entire corpus, we classified the Bugzilla and Github comments using other applicable methods. For example, experiment 4 classified Bugzilla and Github comments according to the *question mark method* and newsgroup articles according to the *RE method*. In these cases, the accuracy scores are copied in grey.

Heuristics that employ *question words* exceed the majority class baseline for Bugzilla comments, while no method exceeds it for GitHub comments. For newsgroup articles, the *question mark* and *RE method* exceed the baseline. On the entire corpus, the highest accuracy, 73.59%, is achieved by the *question mark method* for Bugzilla and GitHub and the *RE method* for newsgroups after *Cleaning*. The improvements over the baselines are statistically significant, according to Fisher's, McNemar's and Chi-square tests.

The best performing heuristic on Bugzilla comments is *question words* with *Cleaning* as a preprocessing step. However, the score is only slightly better than the scores achieved by other methods that use *question words*. In contrast, the same methods tested on the Github comments lead to more variable results. The best accuracy achieved by the *question mark method* after *Cleaning* does not exceed the major class baseline. For newsgroup articles, methods that employ question marks exceed the major class baseline. The highest accuracy is achieved by methods that consider subject line prefixes.

In our supervised experiments, initially we trained machine learners with heuristic features only, and then we included unigram features. Experiments with heuristic features considered 3 classifiers, 63 feature settings and included or excluded *Cleaning* (378 experiments in total, as discussed in section 5.1.). Then, our experiments that included unigram

#	feature value constraints	number of experiments	Accuracy(%)		
			min	avg	max
1	all experiments	63882	66.21	72.89	78.16
2	**Cleaning**	31941	66.21	**73.41**	78.16
3	no Cleaning	31941	67.48	72.36	77.67
4	include unigrams	63504	67.48	72.89	78.16
5	exclude unigrams	378	66.21	71.50	74.18
6	**SVM, linear kernel**	21294	67.48	**73.48**	78.16
7	SVM, RBF kernel	21294	66.21	72.19	77.57
8	RandomForest	21294	67.48	72.99	77.67
9	**omit no heuristic feature**	1014	69.42	**73.78**	77.77
10	#2 + #6 + #9	169	71.17	74.90	77.77
11	#2 + #4 + #6 + #9	168	71.17	74.90	77.77
12	#2 + #5 + #6 + #9	1	73.98	73.98	73.98
13	#2 + #6 + #9 + noun unigram features	42	72.04	75.41	77.77
14	#2 + #6 + #9 + noun and adjective unigram features	42	72.14	75.35	77.77
15	#2 + #6 + #9 + noun, adjective and verb unigram features	42	72.33	74.81	77.48
16	#2 + #6 + #9 + unigram features of all parts-of-speech	42	71.17	74.04	77.57
17	**#2 + #6 + #9 + frequency feature values for unigrams**	56	74.18	**76.46**	77.77
18	#2 + #6 + #9 + binary frequency feature values for unigrams	56	71.94	74.30	77.18
19	#2 + #6 + #9 + tf-idf frequency feature values for unigrams	56	71.17	73.96	76.21
-	*Baseline:* Major class				70.29
-	*Baseline:* Best Unsupervised Method (table 5)				73.59

Table 6: Minimum, maximum and average accuracy achieved by experimental settings that contain the corresponding feature value. Numbers in constraints refer to the inclusion of other constraints mentioned earlier. The last rows reports the major class baseline and the best accuracy achieved by heuristic methods.

features also considered 3 feature value types, 4 parts-of-speech and 14 frequency settings (63,504 experiments in total). The experiment that achieved the highest accuracy, 78.16%, used the linear SVM kernel and included heuristic features for methods: *question mark*, *question mark or question words* and *RE*. It used frequency values for noun and adjective unigram features of frequency 1 or higher after the *Cleaning* preprocessing step.

Although the best accuracy was achieved excluding some heuristic features, this is not the case on average. Experiments where no heuristic features were omitted achieved on average $\sim$0.8% higher accuracy than experiments that omitted heuristic features.

Since it is impractical to present all experimental results, we attempt to judge the importance of feature values by observing the average accuracy achieved by experiments that used each. The unigram frequency threshold does not affect results significantly, since average accuracies per frequency threshold value lie in $[72.62\%, 73.47\%]$.

Table 6 shows the minimum, average and maximum accuracy achieved for various experiment sets. Each set is defined by feature value constraints shown in the second column. For example, set #1 refers to all experiments, i.e. no constraint is applied, set #2 refers to all experiments that considered the *Cleaning* preprocessing step, and set #10 refers to the cut of sets #2, #6 and #9. The most useful feature values are in bold. Sets #2 to #9 show that *Cleaning* is a useful preprocessing step. Equally useful is the inclusion of all heuristics and choosing the *SVM* classifier with linear kernel. Making this selections, sets #10 to #19 show that the next important selection is to include unigram features encoded as frequency feature values. The improvement achieved over the baseline and the heuristic methods is statistically significant.

5.3. Discussion

The main experimental finding is that simple content-based textual characteristics are useful in classifying a communication message as request or non-request. Using them as unsupervised classifiers provides encouraging classification accuracy without any training.

Machine learners trained on these heuristics and unigrams can improve performance by approximately 3%, without any particular feature selection. However, this improvement comes with the cost of annotating communication messages manually for training purposes. The increase requires no costly feature selection, apart from selecting an *SVM* classifier with linear kernel, including unigrams represented as frequency feature values and taking into account all heuristics including *Cleaning* as a preprocessing step. Choosing the exact best performing setting can improve accuracy further up to 1.5%.

Definitely, the actual accuracy levels depend on the communication messages included in the corpus. However, since the selection was performed entirely at random, not associated with threading information, the sample accurately represents the population.

6. Conclusion and future work

In this paper we addressed the task of classifying online newsgroup articles or bug tracking system comments in the domain of *Open Source Software (OSS)* as requests or non-requests. We consider messages that raise a new issue to an

OSS community as requests and all other messages as non-requests. This binary classification is driven by the overall target of assessing the quality of support offered by these means of communication.

We presented a corpus of $1,030$ communication messages selected randomly from NNTP newsgroups and bug tracking systems: Bugzilla and GitHub and annotated manually as requests or non-requests. We identified several heuristics that correlate with this classification and evaluated how well they can perform classification as independent criteria. Then, we combined them together in a feature set to train machine learning classifiers. We evaluated a very wide range of parameters concerning these heuristic features, different machine learners, and unigram features of various parts-of-speech and frequencies. We conclude that heuristic features can perform well for this task, but accuracy needs to be improved for real-life applications. As expected, machine learners can perform significantly better than heuristic based methods.

In the future, we plan to examine if using observations related to the position of a message in a thread can increase accuracy. Usually, threads start with a request. Knowing who submitted the initial request can be useful for classifying messages of that user in the same thread. In addition, it is common that long time after a problem or bug is addressed, some user submits another relevant request. We can also consider other observations that are not based on the content of messages. For example, the fact that frequent users are more likely to be developers and to contribute non-requests than requests. Moreover, often some user submits another relevant request a long time after a problem or bug has been addressed.

Acknowledgements

This work was funded by the European Community's Seventh Framework Program (FP7/2007-2013) [grant number 318736 (OSSMETER)].

7. References

Bagozzi, R. P. and Dholakia, U. M. (2006). Open source software user communities: A study of participation in linux user groups. *Management Science*, 52(7):1099+.

Baldwin, T., Martinez, D., and Penman, R. B. (2007). Automatic thread classification for Linux user forum information access. In *Proceedings of the 12th Australasian Document Computing Symposium (ADCS 2007)*, pages 72–79, Melbourne, Australia.

Baldwin, T., Martinez, D., Penman, R. B., Kim, S. N., Lui, M., Wang, L., and MacKinlay, A. (2010). Intelligent linux information access by data mining: the ILIAD project. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics in a World of Social Media*, WSA '10, pages 15–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bianco, V., Lavazza, L., Morasca, S., Taibi, D., and Tosi, D. (2010). An investigation of the users' perception of OSS quality open source software: New horizons. In Ågerfalk, P., Boldyreff, C., González-Barahona, J. M., Madey, G. R., and Noll, J., editors, *Open Source Software: New Horizons*, volume 319 of *IFIP Advances in Information and Communication Technology*, chapter 2, pages 15–28. Springer Boston, Berlin, Heidelberg.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Crowston, K. and Howison, J. (2005). The social structure of free and open source software development. *First Monday*, 10(2).

Dahlander, L. and Magnusson, M. G. (2005). Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms. *Research Policy*, 34(4):481–493.

Ding, S., Cong, G., Lin, C. Y., and Zhu, X. (2008). Using conditional random fields to extract contexts and answers of questions from online forums. In *Proceedings of ACL-08: HLT*, Columbus, Ohio, USA. Association for Computational Linguistics.

Fuggetta, A. (2003). Open source software–an evaluation. *Journal of Systems and Software*, 66(1):77–90, April.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Exploration Newsletter*, 11(1):10–18.

Hassan, A., Qazvinian, V., and Radev, D. (2010). What's with the attitude? identifying sentences with attitude in online discussions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1245–1255, Cambridge, MA. Association for Computational Linguistics.

Kim, S. N., Wang, L., and Baldwin, T. (2010). Tagging and linking web forum posts. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 192–202, Stroudsburg, PA, USA. Association for Computational Linguistics.

Krishnamurthy, S. (2005). Cave or community?: An empirical examination of 100 mature open source projects. *Social Science Research Network Working Paper Series*.

Lakhani, K. R. and von Hippel, E. (2003). How open source software works: free user-to-user assistance. *Research Policy*, 32(6):923–943.

Lui, M. and Baldwin, T. (2009). You are what you post: User-level features in threaded discourse. In *Proceedings of the 14th Australasian Document Computing Symposium (ADCS 2009)*, Sydney, Australia.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

Samoladas, I., Gousios, G., Spinellis, D., and Stamelos, I. (2008). The SQO-OSS quality model: Measurement based open source software evaluation. In Russo, B., Damiani, E., Hissam, S., Lundell, B., and Succi, G., editors, *Open Source Development, Communities and Quality*, volume 275 of *IFIP International Federation for Information Processing*, pages 237–248. Springer Boston.

Scacchi, W. (2007). Free/open source software development: recent research results and emerging opportunities. In *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium*

on the foundations of software engineering: companion papers, ESEC-FSE companion '07, pages 459–468, New York, NY, USA. ACM.

Spinellis, D., Gousios, G., Karakoidas, V., Louridas, P., Adams, P. J., Samoladas, I., and Stamelos, I. (2009). Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science*, 233:5–28.

Wang, Y.-C. and Rosé, C. P. (2010). Making conversational structure explicit: Identification of initiation-response pairs within online discussions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 673–676, Los Angeles, California. Association for Computational Linguistics.

Wang, L., Kim, S. N., and Baldwin, T. (2010). Thread-level analysis over technical user forum data. In *Proceedings of the Australasian Language Technology Association Workshop 2010*, pages 27–31, Melbourne, Australia.

Wang, L., Lui, M., Kim, S. N., Nivre, J., and Baldwin, T. (2011). Predicting thread discourse structure over technical web forums. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 13–25, Edinburgh, Scotland, UK. Association for Computational Linguistics.