# Pattern-Based Mining of Opinions in Q&A Websites

Bin Lin*, Fiorella Zampetti†, Gabriele Bavota*, Massimiliano Di Penta†, and Michele Lanza*
*Software Institute, Università della Svizzera italiana (USI), Switzerland — †University of Sannio, Italy

***Abstract*—Informal documentation contained in resources such as Q&A websites (*e.g.,* Stack Overflow) is a precious resource for developers, who can find there examples on how to use certain APIs, as well as opinions about pros and cons of such APIs. Automatically identifying and classifying such opinions can alleviate developers' burden in performing manual searches, and can be used to recommend APIs that are good from some points of view (*e.g.,* performance), or highlight those less ideal from other perspectives (*e.g.,* compatibility). We propose** POME **(Pattern-based Opinion MinEr), an approach that leverages natural language parsing and pattern-matching to classify Stack Overflow sentences referring to APIs according to seven aspects (*e.g.,* performance, usability), and to determine their polarity (positive *vs* negative). The patterns have been inferred by manually analyzing 4,346 sentences from Stack Overflow linked to a total of 30 APIs. We evaluated** POME **by (i) comparing the pattern-matching approach with machine learners leveraging the patterns themselves as well as n-grams extracted from Stack Overflow posts; (ii) assessing the ability of** POME **to detect the polarity of sentences, as compared to sentiment-analysis tools; (iii) comparing** POME **with the state-of-the-art Stack Overflow opinion mining approach, Opiner, through a study involving 24 human evaluators. Our study shows that** POME **exhibits a higher precision than a state-of-the-art technique (Opiner), in terms of both opinion aspect identification and polarity assessment.**

## I. Introduction

Online discussions among software developers through various communication channels — *e.g.,* mailing lists, issue trackers, and above all Question & Answer (Q&A) forums such as Stack Overflow— are playing a major and increasing role in software development. Such sources bring various pieces of information, including examples of how to use programming language constructs, APIs or frameworks, and discussions about design choices or algorithmic solutions to certain development problems. To cope with the limited search capabilities of Q&A forums and other forges, and to alleviate developers' burden of manually searching for relevant information, researchers have proposed a wide variety of recommender systems. Such recommending systems can for example link Stack Overflow discussions to code snippets [31], produce documentation [46], enhance existing documentation by mining Stack Overflow discussions [38], or identify insights about APIs [42].

Naturally, developers' discussions contain opinions, *e.g.,* whether a certain API is suitable for solving a given problem, or what the pros and cons of a given framework are. For example, some developers might recommend an API for its rich functionality, while others may warn about its performance. Recommenders could therefore exploit such opinions — *i.e.,* perform *opinion mining* — and suggest APIs that best satisfy the developers' needs, which can be better functionality, better performance, increased compatibility, ease of use, etc.

In Natural Language Processing (NLP), opinion mining has been used in various contexts (*e.g.,* e-commerce, movie streaming [35]) to analyze users' moods and feelings about a given product, expressed in a review written in natural language.

Sentiment analysis [26] is a frequently used opinion mining technique. Its goal is to identify affective states and subjective opinions reported in sentences. In its basic usage scenario, sentiment analysis is used to classify customers' written opinions as negative, neutral, or positive. Sentiment analysis has been used in software engineering for various purposes, such as assessing the polarity of apps' reviews [12], [14], [27], developers' distress or happiness [25], [34], [41], or identifying negative opinions about APIs [47].

Recent work has shown that out-of-the-box sentiment analysis tools are particularly unreliable (and very often in disagreement) when applied to software engineering corpora [20]. Even customized tools (*e.g.,* `SentiStrength-SE`), or retrained tools [23] produced results inadequate in practice for tasks such as API opinion mining. A recent work by Uddin and Khomh [45] dealt with API opinion mining by relying on a keyword-matching approach with a customized Sentiment Orientation algorithm [17].

Stemming from the positive and negative results highlighted in previous attempt to automatically mine API opinions and from the seminal work by Uddin and Khomh [45] in this field, we propose a novel approach named POME which leverages linguistic patterns contained in Stack Overflow sentences referring to APIs, and classify whether (i) a sentence refers to a particular API aspect (functional, documentation, community, compatibility, performance, reliability, or usability), and (ii) it has a positive or negative polarity. The main idea behind POME is to identify whether an API-relevant sentence from Stack Overflow discussions matches any of the 157 manually defined patterns. Each pattern consists of a natural language parse tree where each leaf can either be a generic part-of-speech (*e.g.,* a noun) or, in some cases, a specific part-of-speech (taken from a thesaurus we have built), characterizing an aspect positively or negatively. We have evaluated our approach along three dimensions:

1) We assess the precision and recall of POME in identifying API-related opinions in Stack Overflow on a manually labeled dataset of 1,662 sentences. We compare different variants of POME based on simple pattern matching as well as on machine learning algorithms, finding that its best configuration achieves a precision ranging between 0.61 and 1.00 and a recall ranging between 0.13 and 0.44, depending on the quality aspect subject of the opinion.

2) We compare the performance of the opinion polarity assessment when using pattern matching with six sentiment analysis tools, finding that the defined 157 patterns help in achieving higher values of precision/recall both for positive (0.92 precision and 0.99 recall) and for negative (0.94 precision and 0.73 recall) opinions.
3) We conducted a survey with 24 CS students and professional developers to collect their assessment about the precision of the opinions mined by POME and by the state-of-the-art opinion mining tool Opiner [45] for four popular APIs. The achieved results show that, for most of the quality aspect categories (*e.g.,* usability), POME is able to mine opinions with a higher precision than Opiner.
4) We release POME's source code, the Web app used to label patterns, and the list of patterns we manually defined and all the data used in our evaluations in a replication package [22].

## II. Related Work

We detail related work concerning (i) recommendation of documentation, and (ii) applications of sentiment analysis and opinion mining for software engineering problems.

### A. Recommendation of (formal and informal) documentation

Treude and Robillard [42] proposed an approach to identify API (library) insights in Stack Overflow. We detail this work in Section III since POME uses a (slightly modified) reimplementation of this approach for tracing Stack Overflow sentences onto APIs. However, POME also provides positive or negative opinions about a specific aspect of the library.

Wong *et al.* [46] proposed AutoComment to mine comments from Stack Overflow and automatically describe source code similar to snippets discussed on Stack Overflow. Differently from AutoComment, we focus on mining opinions about APIs rather than code snippet descriptions. Chatri and Robillard [32] developed an approach for identifying relevant portions of documentation to support developers seeking information. Other work ([11], [21], [33], [37]) focused on suggesting relevant documents, discussions and code samples from the Web to fill the gap between the IDE and the Web browser. Subramanian *et al.* [38] proposed Baker to enhance API documentation by linking to it relevant code examples. Among the various resources available on the Web, Q&A Websites — and in particular Stack Overflow— have been the basis of many recommender systems [8], [29], [30], [39]. Our approach is an add-on to these recommenders, as it could be used to tag APIs in recommended snippets with quality badges mined from developers' opinions.

### B. Sentiment analysis/opinion tools and their application to software engineering problems

Sentiment analysis has been used in various areas of SE, for example to analyze the sentiment of commit comments in GitHub [14], the correlation between the sentiment in 560k JIRA comments and the time to fix a JIRA issue [25], or how the sentiment of developers is affected by the result of a build process in continuous integration [36].

Besides the analysis of developers' behaviors, sentiment analysis has been used to analyze users (or developers) opinions about software applications. Specifically, Guzman *et al.* used sentiment analysis to classify tweets related to software projects [13]. Several authors have used sentiment analysis to support the evolution of mobile applications [6], [12], [15], [27], by prioritizing user reviews based on the expressed sentiment.

The most adopted sentiment analysis tool in SE is SentiStrength [40], which is based on a sentiment word strength list plus some heuristics including spell checking and negation handling. Its word lists are based on comments taken from myspace.com/, making it unsuitable for SE applications.

NLTK [18] is a lexicon and rule-based sentiment analysis tool leveraging VADER (Valence Aware Dictionary and sEntiment Reasoner), which is tuned to social media text (especially micro-blogging). Differently, Stanford CoreNLP [35] leverages a Recursive Neural Network (RNN) and is able to compute the sentiment of a sentence based on how words compose the meaning of the sentence, and not by summing up the sentiment of individual words. However, Stanford CoreNLP was trained on movie reviews, a non-SE domain.

Since most existing sentiment analysis tools were not conceived to be applied on SE artifacts, researchers posed questions on their applicability in this domain. Tourani *et al.* [41] found that SentiStrength achieves a very low precision when analyzing mailing lists. Novielli *et al.* [24] also highlighted the challenges of employing sentiment analysis techniques to assess the affective load of text containing technical lexicon. Jongeling *et al.* [20] conducted a comparison of four widely used sentiment analysis tools and found none of them can provide accurate predictions in the SE domain.

To overcome limitations of existing tools, Islam and Zibran [19] developed SentiStrength-SE, which includes word lists and heuristics specific for SE documents. Calefato *et al.* proposed EmoTxt [5] to recognize specific emotions in SE datasets, such as Stack Overflow and JIRA. Also, Calefato *et al.* developed Senti4SD [4], a sentiment analysis tool trained on Stack Overflow and leveraging lexicon and keyword-based features as well as word embedding.

de la Mora and Nadi [9] proposed the use of metric-based comparison to support developers in selecting the library to use. Our work aims at mining API opinions from Stack Overflow, thus being complementary to the quantitative metrics used in [9].

The closest work to ours is Opiner by Uddin and Khomh [45], able to detect the polarity of sentences related to APIs by customizing the Sentiment Orientation algorithm [17]. The algorithm was originally developed to mine customers' opinions on computer products. Uddin and Khomh customized the tool with words specific to API reviews. Opiner classifies the mined opinions into "aspects" by exploiting machine learning classifiers using the frequencies of single words and n-grams appearing in the sentences as predictor variables.

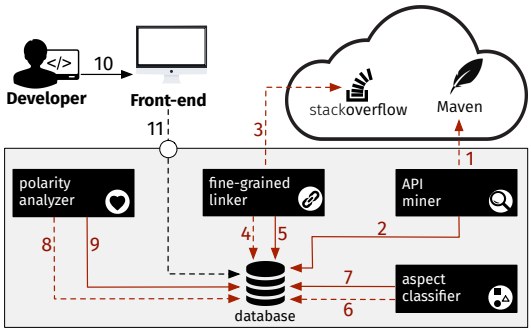



Fig. 1. The POME architecture.

We took inspiration from this work. Indeed, the goal of POME is the same of Opiner. We aim at automatically mine opinions about APIs in Stack Overflow discussions and classify them based on their aspect and sentiment polarity. However, we show that a simpler pattern-based approach can provide a substantial step ahead in the accuracy of the opinions mined from Stack Overflow. We also show that pattern matching is more precise than sentiment analysis tools for identifying the polarity of opinionated sentences.

## III. POME

Fig. 1 depicts POME's architecture. The dashed arrows represent dependencies while full arrows indicate flows of information pushed from one component to another. Arrows depicted in red indicate operations performed at the beginning and then refreshed periodically with the goal of storing crowdsourced opinions about APIs; the black arrows represent actions triggered by a POME user from the front-end.

The *API miner* extracts all available Java APIs from the Maven central repository [1] (① in Fig. 1). We select for each API its: (i) name, (ii) description, (iii) link to the `jar` of the latest version, and (iv) release date of the `jar`. We collected this information for a total of 116,318 APIs, between May and June 2017, storing it in our database ②.

The *fine-grained linker* mines Stack Overflow discussions to establish links between the APIs stored in the database ④ and relevant sentences in Stack Overflow discussions ③. For example, the sentence "*Apache commons-io is the straightforward solution to programmatically copy files*" is linked to the `commons − io` library by using an approach built on top of the one proposed by Treude and Robillard [42].

Knowing the sentences related to an API, the *aspect classifier* categorizes the semantic content of each sentence in one of the eight aspects described in Table III ⑥, and adds this information to the database ⑦. The sentences not classified as "none" (*i.e.,* those discussing quality aspects relevant to mined opinions about APIs) are then analyzed by the *polarity analyzer* ⑧, that identifies the sentiment they express and consequently their polarity, *i.e., positive* or *negative* (we ignore sentences having a *neutral* sentiment since they are not of interest when mining opinions), and stores this information in the database ⑨.

To use POME, a developer interested in accessing opinions about an API can submit a textual query through the Web-based front-end ⑩. She can search for a specific API or, if she does not know which API to use, the query can be used to describe the task she wants to perform (*e.g.,* reading `JSON` files in Java). This information is provided to a Web service ⑪ to identify the most relevant APIs for the given query and provide as output the opinions mined for them.

In the following, we detail the main POME components. The *API miner* is not described since it is a simple Python Web scraper to mine the Maven central repository.

### A. Fine-grained Linker

This component retrieves sentences from Stack Overflow posts related to a given API. Given an API (*e.g.,* `GoogleGson`), we use the information collected by the *API miner* to download its `jar`. Using Java Reflection we extract the complete list of its classes and methods and link sentences in Stack Overflow discussions to APIs, using a reimplementation of the linker by Treude and Robillard [42]. There are two differences between our approach and the one by Treude and Robillard [42]. First, while they use the Stack Overflow API to retrieve the Stack Overflow discussions, we rely on the December 2017 official Stack Overflow data dump to avoid issues related to usage limitations of the API. Second, they use the first three regular expressions reported in Table I to identify sentences including (i) the fully-qualified API type (*e.g.,* `com.google.code.gson`); (ii) the non-qualified API type (*e.g.,* `Gson`); and (iii) the link to the official API documentation (*e.g.,* https://sites.google.com/site/gson/gson-user-guide). In our approach, we also retrieve Stack Overflow sentences matching the fourth regular expression shown in Table I. We decided to include this fourth regular expression since we observed that many sentences on Stack Overflow discuss issues related to APIs by referring to specific APIs rather than to the API type (*i.e.,* name) or to its documentation. While this additional regular expression might introduce false positives, matching both the class name and the method name mitigates this risk. We discuss the precision of this additional regular expression in Section VI.

We use the *fine-grained linker* to identify all relevant sentences for a given API only from Stack Overflow answers (*i.e.,* we do not consider questions), because opinions are unlikely to reside in the questions, where users mostly ask for help. Also, we discard sentences belonging to questions posted before the release date of the library `jar` under analysis, to reduce the risk of mining opinions referring to old releases of the library. The sentences identified by the *fine-grained linker*, along with the link to the respective API, are stored in the POME's database for all previously mined APIs.

### B. Aspect Classifier

The *aspect classifier* analyzes the stored sentences to identify the quality aspect(s) discussed in them. In the following, we discuss different ways to perform this task, while in Section IV we explain how we identified the best solution.

TABLE I
REGULAR EXPRESSIONS FOR EXTRACTING API-RELATED SENTENCES IN STACK OVERFLOW ANSWERS.

| Regular expression | Description | Is case sensitive? |
|---|---|---|
| $(?i).*\backslash bPackageName\backslash.TypeName\backslash b.*$ [42] | Fully-qualified API type | |
| $.*(\mid$ [a-z]+ $\mid[\backslash.!?]\mid[\backslash(<])TypeName)([> \backslash)\backslash.,!?\$]\mid$[a-z]+$).*$ [42] | Non-qualified API type | ✓ |
| $.* < a.*$href$.*PackageName/TypeName\backslash.html.* > .* < /a > .*$ [42] | Link to the API official documentation | ✓ |
| $.*ClassName\backslash.MethodName[\backslash(\mid ]$ | Reference to a method of a specific class | ✓ |

TABLE II
DATASET USED FOR PATTERNS' DEFINITION AND TRAINING OF THE MACHINE LEARNING ALGORITHMS.

| Category | # sentences linked | # sentences validated | URL |
|---|---|---|---|
| Bytecode APIs | 2,645 | 999 | goo.gl/rzoqc7 |
| Embedded SQL DB | 622 | 622 | goo.gl/kknzvD |
| HTTP Clients | 1,714 | 999 | goo.gl/b8vgQN |
| JSON APIs | 4,764 | 999 | goo.gl/9cas1C |
| Reflection APIs | 481 | 481 | goo.gl/6935xc |
| SSH APIs | 246 | 246 | goo.gl/2ih4h6 |
| **Overall** | **10,481** | **4,346** | **-** |

*1) Pattern matching:* The conjecture is that users providing opinions about APIs on Stack Overflow tend to use repetitive discourse patterns that can be encoded to capture both the quality aspect(s) and the sentiment of the opinion (thus, pattern matching can be used in the context of the *polarity analyzer*). To identify the patterns, we manually analyzed 4,346 Stack Overflow sentences identified by the *fine-grained linker* as related to APIs belonging to the six categories of popular APIs (provided by Maven central) reported in Table II.

Table II reports the name of the category, the number of API-related sentences extracted from Stack Overflow discussions, the number of sentences we manually analyzed, and the link to Maven central listing the APIs belonging to the specific category. From each category, we only extracted sentences related to the five most used APIs listed on https://mvnrepository.com/. For categories having more than 1,000 linked sentences, we manually analyzed only a randomly selected subset to avoid bias in the definition of the patterns (*i.e.,* extract patterns that are very specific to one predominant API category in our dataset).

The 4,346 sentences have been manually analyzed by four of the authors (from now on evaluators) to categorize each one as expressing or not an opinion about the linked API. Each sentence was randomly assigned to two of the four evaluators, resulting in $\simeq$2,180 sentences per evaluator. In case a sentence did not report any opinion, we assigned the "*none*" label. If an opinion was identified, the evaluator firstly selected the part of the sentence reporting the opinion. Then, she classified the selected part of the sentence in terms of the quality aspect(s) the opinion refers to (*e.g., compatibility*). No predefined list of quality aspects was provided. However, every time the evaluator had to analyze a sentence, the Web application showed the list of quality aspects created so far, allowing the evaluator to select one of the already defined aspects. In a context like the one encountered in this work, where the number of possible quality aspects might be large, such a choice helps using consistent naming without introducing a substantial bias. Table III presents the list of aspects obtained during the labeling process.

The evaluator also assigned a *negative* or *positive* sentiment to the reported opinion (this information is used in the context of the *polarity analyzer*) and, finally, she identified in the selected part of the sentence the Parts-of-Speech (POS) referring to the linked API and the quality aspect(s), *i.e.,* noun, adjective, etc. To better understand the process, let us discuss an example of manual analysis. Consider the sentence: *"Based on my personal experience, Gson is the fastest library out there"*. First, the evaluator selects the part reporting the opinion, in this case: "*Gson is the fastest library*". Then, she assigns the *performance* quality aspect and a *positive* sentiment to it. Finally, she marks "Gson" as a proper noun referring to the library, and "fastest" as an adjective related to the quality aspect assigned to the opinion (*i.e., performance*).

Once each sentence was manually analyzed by any two of the evaluators, we solved all conflicts by adding a third evaluator not previously involved in the analysis of that sentence. A conflict could be related to (i) the part of the sentence selected as opinion, (ii) the sentiment assigned to the opinion, and (iii) the quality aspect(s) identified.

The output includes 388 sentences classified as reporting an opinion and referring to seven different quality aspects (and 3,958 discarded as not discussing quality aspects). Table III reports the number of positive and negative opinions identified for each of them. About 9% of the linked sentences (388/4,346) explicitly report negative or positive opinions related to one of the quality aspects. While the percentage might look low, if we consider the number of posts on Stack Overflow ($\sim$50M at the date of the writing), the amount of opinions is still impressive.

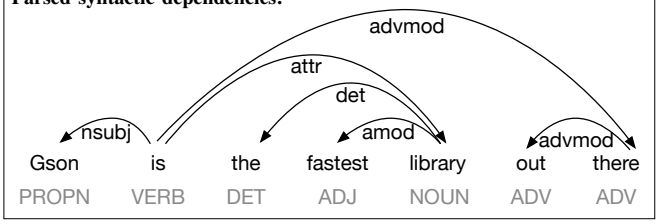
The 388 API-related sentences manually annotated have been exploited to identify recurrent patterns used in Stack Overflow discussions for expressing opinions about APIs. With "patterns" we refer to lexical rules that capture the syntax and semantic of the opinionated sentences. One of the evaluators conducted a pilot study using API-related sentences including opinions about *performance*. Since we wanted to define patterns considering both the syntax and the semantic of the API-related sentences, the evaluator working on the patterns' extraction not only had the quality aspect and the sentiment assigned to each sentence as information, but also the parts of speech related to each token (*i.e.,* noun, verb, adjective, adverb etc.) as well as their syntactic dependencies.

To reduce the number of patterns belonging to the same quality aspect, the evaluator could also create a bag of words related to verbs, adjectives and adverbs and use them for defining patterns. A positive pattern belonging to the *performance* category is shown in the following.

TABLE III
ASPECTS USED BY THE ASPECT CLASSIFIER AND SENTENCES IDENTIFIED FOR EACH OF THEM DURING MANUAL ANALYSIS.

| Quality Aspect | The sentence talks about... | # Opinions | |
|---|---|---|---|
| | | Negative | Positive |
| Community | The activities of the community maintaining the API (*e.g.,* is the API actively maintained?) | 2 | 8 |
| Compatibility | The compatibility of the API with respect to specific platforms, programming languages, or other APIs | 21 | 10 |
| Documentation | The content/quality of the API documentation | 3 | 29 |
| Functional | The features offered/not offered by the API | 13 | 153 |
| Performance | The performance of the API (*e.g.,* speed, memory footprint) | 12 | 26 |
| Reliability | The reliability of the API (*e.g.,* whether it is buggy or not) | 18 | 10 |
| Usability | The usability of the API, in terms of how easy is to use/adapt it and evolve/maintain the code using it | 9 | 74 |
| *None* | *None of the above-listed aspects* | *3,958* | |

**Quality Aspect & Sentiment:** Performance | Positive
**Rule:** [Verb_To_Be] [Pos_Adjective_Performance]
**Dependency requirement:** [Verb_To_Be] should be the first parent node of [Pos_Adjective_Performance] with a POS tag of verb.
**Example:** Gson [*is*] the [*fastest*] library out there.
**Parsed syntactic dependencies:**




The *Pos_Adjective_Performance* includes positive adjectives linkable to performance, such as *fastest*, *performant*, etc.

Once the pilot study was completed, the evaluator trained other three evaluators in a 30-minute session that involved discussing the results and some ambiguous sentences. The API-related sentences belonging to the other six quality aspects were randomly distributed among the four evaluators. For each quality aspect, all the API-related sentences were coded by the same evaluator. The same API-related sentence can fall into more than one quality aspect. For this reason, it is possible to infer more than one pattern from the same sentence. At the end of the patterns' extraction, all the evaluators created a catalog of inferred patterns to merge similar patterns into a more general pattern. Each decision taken at this stage was representative of the opinion of all evaluators.

In the end, we obtained a list of 157 patterns, each one representative of a specific quality aspect expressing a specific sentiment. Given a sentence $S$ as input, the *aspect classifier* can then be used to check whether $S$ matches one of the defined patterns. To do this, the *aspect classifier* uses the `spaCy` [2] NLP library to build a dependency tree of $S$. The tree reports (i) the POS in $S$, and (ii) the dependency relations between the tokens composing $S$. This allows to (i) easily verify whether $S$ matches a given pattern, and (ii) identify negated terms, needed to correctly assess the sentiment polarity of the matched pattern (*e.g.,* if a positive pattern for performance is matched but a positive performance adjective is negated, then the sentiment polarity is inverted to negative).

*2) Machine learning:* Another possibility to implement the *aspect classifier* is to use a machine learning algorithm trained on a set of manually labeled sentences.

We exploit previously labeled sentences (Table II) to train machine learners to classify a given sentence into eight categories: the seven quality aspects we consider plus "none". Specifically, we used all the sentences with opinions and randomly selected same amount of sentences without opinions for training to avoid bias. We used the Scikit-learn [28] Python library to experiment with 10 different machine learners. As predictor variables, we used the terms contained in the sentences. For preprocessing we remove stop words and punctuations, and performed word stemming. We considered each term as a predictor variable. Besides analyzing the single words contained in each sentence, we extract the set of *n-grams* composing it, considering $n \in [2 \dots 3]$.

We consider as features for the machine learner the presence/absence of the 157 patterns, *i.e.,* whether a sentence matches each of the patterns we previously defined. There is a key difference between the pattern matching approach and employing patterns as a feature of a machine learner. In the first case, patterns are used as rules, and therefore sentences matching a given pattern are automatically classified into an aspect and sentiment polarity. In the second case, the presence of a pattern, may (or may not) contribute towards a classification along with other features. We experimented each machine learner with seven different combinations of features: (i) BOW-only (Bag Of Words), only considering single terms, (ii) n-grams-only, (iii) patterns-only, (iv) BOW+n-grams, (v) BOW+patterns, (vi) n-grams+patterns, and (vii) BOW+n-grams+patterns.

A possible problem is that some categories are rarer than others. A machine learning algorithm tends to assign sentences to more frequent categories, because an error in under-represented categories is more acceptable than an error in other categories to achieve a better overall accuracy. To prevent this, we re-balanced our training set using SMOTE [7], an oversampling method which creates synthetic samples from the minor class. We experimented each algorithm both with and without SMOTE.

### C. Polarity Analyzer

The *polarity analyzer* analyzes the sentences classified as relevant by the *aspect classifier* to identify the sentiment polarity of the opinions. We investigated two different options for the implementation of the *polarity analyzer*, and we evaluate their performance to pick the best one (see Section IV).

*1) Pattern matching:* The set of 157 patterns we extracted for the *aspect classifier* can be used also to assess the sentiment polarity of the opinions.

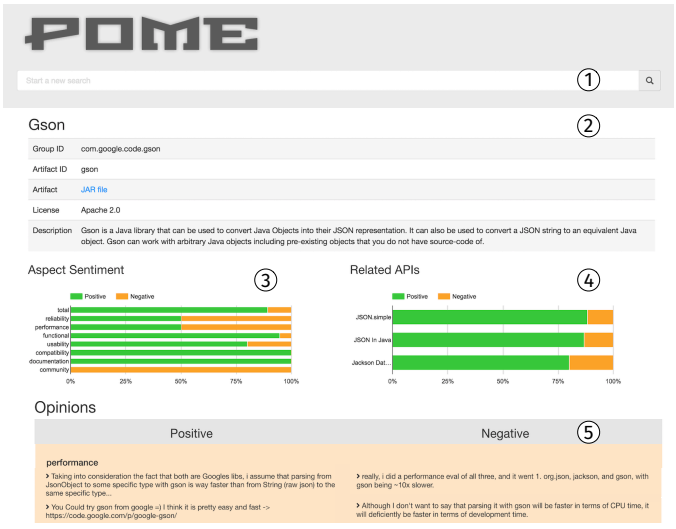



Fig. 2. Information and opinions about the "Gson" API presented by POME.

Indeed, each pattern is related to an aspect and to a sentiment polarity. Thus, the first possibility is to use pattern matching to identify the sentiment of opinions.

*2) Sentiment Analysis Tools:* A second possibility to determine a sentence's sentiment polarity is to exploit one of the many sentiment analysis tools existing in the literature. We experimented with six of them with their default settings: SentiStrength [40], SentiStrength-SE [19], NLTK [18], SentiCR [3], Senti4SD [4], and Stanford CoreNLP [35].

### D. POME *in Action*

We implemented POME as an online application. POME implements a Java API search engine. A developer who needs to parse JSON files without prior knowledge of any relevant API, can search with a query "parse JSON". POME uses Information Retrieval (IR) techniques to list the APIs in the database having a textual description relevant for the query. The developer can select an API, for example "Gson", to assess what the users' opinions about this API are.

POME will then present relevant information about "Gson" as shown in Fig. 2, and including:

1) **Basic information.** The API group ID, artifact ID, link to the `jar` file, license, and description ②.
2) **Opinions on the API classified by aspect.** POME analyzes the polarity of the mined opinions and presents the results with a bar chart ③, where the green and orange depict the percentages of positive and negative opinions, respectively. Each bar in the chart stands for one aspect, while the top bar summarizes the overall polarity of all opinions, that are listed in the table below ⑤. By clicking a bar in the chart, POME only shows in the table opinions related to the aspect of interest.
3) **Opinions on related APIs.** POME also presents a bar chart ④ summarizing opinions of related APIs, *i.e.,* same/similar functionality, identified as the ones having a high textual similarity in terms of description or belonging to same categories in Maven. Each bar stands for one API, and bars are ordered by decreasing ratio of positive opinions. Users can open the information pages of related APIs by clicking the bars.

TABLE IV
Dataset used to answer $RQ_1$ & $RQ_2$.

| Category | # APIs | # sentences | URL |
|---|---|---|---|
| Configuration APIs | 20 | 67 | goo.gl/gnQr51 |
| Mocking | 37 | 199 | goo.gl/6iTVeQ |
| Validation Frameworks | 40 | 171 | goo.gl/sQ15rp |
| XML Processing | 34 | 468 | goo.gl/TwPtgD |
| JDBC Pools | 5 | 757 | goo.gl/yDuWq1 |
| **Overall** | **136** | **1,662** | **-** |

TABLE V
Dataset used to answer $RQ_3$

| Aspect | POME opinions | | | Opiner opinions | | |
|---|---|---|---|---|---|---|
| | # pos | # neg | # sum | # pos | # neg | # sum |
| community | 1 | 0 | 1 | 2 | 0 | 2 |
| compatibility | 6 | 5 | 11 | 3 | 0 | 3 |
| documentation | 16 | 0 | 16 | 9 | 4 | 13 |
| functional | 123 | 10 | 133 | 19 | 13 | 32 |
| performance | 11 | 8 | 19 | 5 | 2 | 7 |
| reliability | 3 | 4 | 7 | 2 | 22 | 24 |
| usability | 16 | 2 | 18 | 92 | 35 | 127 |
| **total** | **176** | **29** | **205** | **132** | **76** | **208** |

## IV. Study Design

The *goal* of this study is to evaluate the accuracy of POME in mining opinions from Stack Overflow discussions and classifying these opinions according to the quality aspects they refer to (*e.g.,* performance, usability) and their sentiment polarity (*i.e.,* negative or positive). The *context* of the study consists of 2,075 sentences extracted from Stack Overflow discussions related to 140 APIs from the Maven central repository. The material used in this evaluation along with its working data set is available in our replication package [22].

### A. Research Questions

We aim at answering the following research questions (RQs):

**$RQ_1$:** *How does a rule-based aspect classifier for Stack Overflow perform, compared to machine learning approaches?* This RQ compares the performance of different implementations of the *aspect classifier*, *i.e.,* the pattern matching approach and the machine learning approaches.

**$RQ_2$:** *How does the rule-based polarity analyzer perform, compared to state-of-the-art sentiment analysis tools?* This RQ evaluates the accuracy of the *polarity analyzer* when using (i) a pattern matching approach, or (ii) six state-of-the-art sentiment analysis tools.

**$RQ_3$:** *How does POME perform compared to Opiner, a state-of-the-art tool for mining opinions from Stack Overflow?* This RQ compares POME with Opiner [45].

### B. Context Selection & Data Collection

Table IV and Table V present the datasets we used.

*1) $RQ_1$ and $RQ_2$:* We considered a set of sentences from Stack Overflow discussions, mined from the official Stack Overflow dump dated Dec 2017, identified using our *fine-grained linker* as relevant to one of the 136 APIs belonging to the five popular categories of APIs from Maven central listed in Table IV. 1,662 sentences were mined as relevant to at least one of the 136 subject APIs. The 136 APIs used in the context of $RQ_1$ and $RQ_2$ have not been used to define the patterns exploited by our approach for the opinions detection and classification. We performed a manual analysis to categorize each of the 1,662 sentences as expressing or not an opinion about the linked API. In case the sentence did not report any opinion, we assigned it to a "*no opinion*" label. Instead, if an opinion was identified, the sentence was further classified in terms of the quality aspect(s) the opinion refers to (*i.e.,* one or more among *community*, *compatibility*, *documentation*, *functional*, *performance*, *reliability*, and *usability*). Finally, the sentiment of the reported opinions was manually assessed by assigning a value between *negative* and *positive*.

The manual analysis was performed by three of the authors and, also in this case, was supported by a Web application ensuring that two authors were assigned to each sentence. All 1,662 sentences were labeled by two authors. The Cohen's kappa coefficient is 0.6492 for sentiment and is 0.6494 for aspect, which demonstrates a substantial agreement. A fourth author not involved in the manual analysis then solved conflicts. A conflict can concern the sentiment of a sentence as well as the quality aspects assigned to it. Overall, 523 sentences (31%) were classified as reporting opinions (505 related to one aspect, 18 to two aspects): *community* (10), *compatibility* (73), *documentation* (41), *functional* (246), *performance* (30), *reliability* (56), and *usability* (85). This manual process was performed before the definition of the patterns' catalog to avoid the authors being influenced during the process. Also, in $\mathbf{RQ}_3$ we involved external evaluators in the judgment of the opinions mined by POME (and by Opiner [45]), to have an external and unbiased view on the quality of the mined opinions.

To answer $\mathbf{RQ}_1$, we ran different POME implementations on the dataset of 1,662 sentences to assess their accuracy in identifying opinion aspects. The implementations include the pattern matching approach and machine learning approaches in all variations presented in Section III.

Concerning $\mathbf{RQ}_2$, we compared the accuracy of POME in assessing the sentiment of opinions with the six sentiment analysis tools mentioned in the previous section. We only conducted the comparison on the subset of 523 sentences for which the best configuration of the *aspect classifier* (output of $RQ_1$) can detect the existence of opinions. Indeed, when envisioning POME as a tool deployed to mine opinions and assign a polarity to them, our priority was to identify the *polarity analyzer* implementation better suited for the sentences identified by the *aspect classifier* as opinions, since the discarded ones are not shown to the POME user.

*2) $RQ_3$:* To compare with Opiner [45], we collected the opinions mined by the two tools for four APIs including "`springframework`", "`glassfish.jersey`", "`mongodb`", and "`google.gwt`", and asked developers and CS students to assess their accuracy. Those APIs are listed in the top-ten "most reviewed APIs" in Opiner [43] and were not used in the POME's pattern definition nor in $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$. Once the best *aspect classifier* ($\mathbf{RQ}_1$) and *polarity analyzer* ($\mathbf{RQ}_2$) were identified, we ran POME on the Stack Overflow data dump to identify opinionated sentences related to the four APIs, collecting in total 205 opinions.

To compare with Opiner we performed the following steps. First, we collected the opinions mined by Opiner for the subject APIs from the original implementation of the authors [43]. Second, we only considered the opinions mined by Opiner for the same APIs that are related to the same aspects used in POME. Third, Opiner uses a set of heuristics to link Stack Overflow sentences onto APIs. One of the heuristics it uses is the explicit mention of the library in the sentence (similar to what we also do). Other heuristics focus on increasing the number of collected opinions (*i.e.,* higher recall) at the expense of precision. For example, the "same conversation association" links an opinionated sentence to the nearest library mentioned in a Stack Overflow conversation. Since in $\mathbf{RQ}_3$ we evaluate the precision of the mined opinions, we did not want to penalize Opiner by considering for POME sentences linked with an approach designed to ensure high precision (like the one implemented in our *fine-grained linker*) and for Opiner sentences linked with heuristics possibly introducing imprecisions. Therefore, among all opinionated sentences mined by Opiner, we only considered those explicitly mentioning the subject library. Finally, since Opiner identified more sentences than POME, we tried to balance the number of sentences to be evaluated by participants for the two tools: if the number of sentences identified by Opiner for a specific aspect was lower or equal than 10, we kept all sentences related to that aspect. This applied to *community*, *compatibility*, and *performance*. Otherwise, for a given aspect $A_i$, we compute the percentage $p_{A_i}$ of sentences identified by Opiner for $A_i$ (*e.g.,* if 10 out of 100 overall opinions mined by Opiner are related to $A_i$, then such a percentage is 10%). Then, we randomly select $p_{A_i} \times n_{POME}$, where $n_{POME}$ is the total number of opinions identified by POME, among those identified by Opiner for $A_i$.

We invited 11 developers, 12 CS students (BSc, MSc, PhD), and 1 postdoc to evaluate the accuracy of the opinions mined by POME and by Opiner for the subject APIs. Participants had an average of 7.5 years of Java development (median=6). Each participant was asked to use a Web app to label the aspect and sentiment polarity (positive, neutral, negative) expressed in the sentences. While the tools automatically classify the sentiment polarity into positive or negative, we gave to the annotators the option to select neutral, to identify false positives in the sentiment identification. The sentences were randomly selected from the considered APIs, and shown in random order. Participants were not aware that the opinions were extracted from different tools to avoid any type of bias. Each sentence was labeled by two participants, and participants were required to label at least 30 sentences. On average, participants labeled 48.5 sentences (median=36).

Each sentence was firstly labeled by two participants. If two participants did not agree with each other on either aspect or sentiment, a third participant would be asked to solve conflicts related to the aspect classification and to the sentiment polarity again through the Web app. For 18 sentences identified by Opiner, the participants solving the conflict were not able to assign an aspect/sentiment with a high confidence. Thus, we preferred to exclude these 18 sentences from our dataset, as they are characterized by a high degree of subjectivity (three humans were not able to agree on the aspect and or sentiment polarity). The final number of opinions evaluated in $RQ_3$ for each tool is reported in Table V.

### C. Data Analysis

To answer $\mathbf{RQ_1}$ we compare the precision and recall of each experimented approach in classifying sentences (as belonging or not to one of the seven aspects) for the dataset of 1,662 sentences. To answer $\mathbf{RQ_2}$ we compare the precision and recall of the sentiment analysis classification performed by the pattern matching approach and the six sentiment analysis tools. To answer $\mathbf{RQ_3}$ we compare the precision of the opinions mined by POME and Opiner both in terms of aspects they identify and sentiment assigned to the opinions. We report the percentage of correctly identified aspects and sentiment for both tools. To compare the precision of POME and Opiner we use Fisher's exact test [10], which statistically compare proportions. Since we perform multiple comparisons (one for each aspect) we adjust $p$-values using Holm's correction [16]. We also report, for the overall dataset, the Odds Ratio (OR) *i.e.,* the ratio between the chance (odd) POME has to correctly classify aspect and sentiment, vs. odd achieved by Opiner.

## V. Results Discussion

**$\mathbf{RQ_1}$: *How does a rule-based aspect classifier for Stack Overflow perform, compared to machine learning approaches?*** Table VI reports the precision and recall in detecting each of the seven quality aspects discussed in API-related sentences. Table VI compares the performance obtained using the *pattern matching* approach (black row) and the best performing machine learner, *i.e., LinearSVM*. We show the results when using SMOTE to balance the training set, since it ensured a boost in performance. Also, we do not show the results when using n-grams only, as this approach obtained poor accuracy. The complete results including all machine learning approaches are in the replication package [22].

While a reasonable recall is useful to get enough recommendations, in the context of opinion mining a high precision is preferable to avoid misleading recommendations to developers.

Using BOW for training the machine learner guarantees a relatively high precision for two of the seven quality aspects, namely *usability* (0.88) and *performance* (0.75), with a recall floating around 0.10. Adding n-grams does not significantly improve the performance of POME with respect to BOW-only. The only exception is for *reliability* for which the LinearSVM is able to reach a precision equals to 1, but with a very low recall (0.02). The limited contribution of n-grams is in line with the findings of Uddin and Khomh [44].

When patterns are included as features (gray rows in Table VI), the performance substantially improves, especially for precision. Moreover, training the LinearSVM with patterns only is sufficient to obtain the similar performance ensured by the combination of all features (BOW+n-grams+patterns). This confirms the pivotal role of patterns in the classification.

Finally, the last row of Table VI reports results obtained using the patterns as rules (*i.e.,* plain pattern matching) without any learning algorithm. The precision for all aspect categories is comparable to the one obtained using patterns as features for training LinearSVM, with the exception that other approaches failed to detect sentences with *community* aspect. It is worth noting that the recall is significantly higher. The approach using pattern-matching is able to obtain, for each quality aspect, a precision varying in the range [0.61-1.00] with a recall varying in [0.13-0.44]. The API-related sentences belonging to *documentation* or *performance* are the ones better identified in terms of both precision (0.95 and 1.00) and recall (0.44 and 0.40). For both *reliability* and *community*, the precision is high (0.78 and 1.00, respectively) with a low recall (0.13 and 0.20).

Given the above results, our decision was to implement the aspect classifier of POME using the pattern matching approach, given its simplicity and performance.

**$\mathbf{RQ_2}$: *How does the rule-based polarity analyzer perform, compared to state-of-the-art sentiment analysis tools?*** To answer $RQ_2$, we use the 186 API-related sentences identified as containing opinions when running the implementation of the aspect classifier chosen in $RQ_1$. Table VII reports the precision and recall, of (i) six state-of-the-art sentiment analysis tools and (ii) the pattern-based approach, in identifying the sentiment expressed in the sentences. As also highlighted in previous literature [23], sentiment analysis tools show poor performance in identifying the sentiment (positive or negative) reported in software engineering datasets. Our results tend to confirm the above statement and, most importantly, underline how the pattern-based approach outperforms the state-of-the-art tools for both positive and negative opinions. This is expected since (i) the patterns have been properly determined looking at API-related sentences mined from Stack Overflow, and (ii) the sentences considered for evaluation have been selected using the approach that verifies the presence of at least one of the 157 patterns. Specifically, for both positive and negative opinions, the pattern-based approach has a precision $\geq 0.90$. The recall is higher for positive opinions than for negative ones (0.99 and 0.64 respectively).

To sum up, the pattern-based approach has good performance in terms of both precision and recall, while for sentiment analysis tools a high precision comes at the expense of low recall. The only exception to this trend is *Standford CoreNLP* that, however, exhibit a very low precision for the negative opinions. Looking more in-depth at the low recall of sentiment analysis tools, it is possible to state that the big challenge resides in the presence of many sentences wrongly classified as neutral.

TABLE VI
PERFORMANCE OF THE BEST MACHINE LEARNING APPROACH USING SEVEN DIFFERENT SET OF FEATURES AND THE PATTERN MATCHING APPROACH.

| | Community | | Compatibility | | Documentation | | Functional | | Performance | | Reliability | | Usability | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pr | Rc | Pr | Rc | Pr | Rc | Pr | Rc | Pr | Rc | Pr | Rc | Pr | Rc |
| **BOW-only** | 0.00 | 0.00 | 0.39 | 0.10 | 0.21 | 0.71 | 0.33 | 0.03 | 0.75 | 0.10 | 0.21 | 0.09 | 0.88 | 0.08 |
| **BOW+n-grams** | 0.00 | 0.00 | 0.38 | 0.07 | 0.34 | 0.45 | 0.26 | 0.11 | 0.67 | 0.07 | 1.00 | 0.02 | 1.00 | 0.08 |
| **patterns-only** | 0.00 | 0.00 | 0.75 | 0.12 | 1.00 | 0.21 | 0.63 | 0.10 | 1.00 | 0.37 | 0.00 | 0.00 | 1.00 | 0.13 |
| **BOW+patterns** | 0.00 | 0.00 | 0.76 | 0.30 | 0.60 | 0.36 | 0.66 | 0.16 | 1.00 | 0.37 | 0.00 | 0.00 | 1.00 | 0.13 |
| **n-grams+patterns** | 0.00 | 0.00 | 0.75 | 0.12 | 1.00 | 0.21 | 0.63 | 0.10 | 1.00 | 0.37 | 0.00 | 0.00 | 1.00 | 0.13 |
| **BOW+n-grams+patterns** | 0.00 | 0.00 | 0.87 | 0.27 | 1.00 | 0.24 | 0.63 | 0.13 | 1.00 | 0.37 | 0.00 | 0.00 | 1.00 | 0.13 |
| **Pattern matching** | 1.00 | 0.20 | 0.86 | 0.33 | 0.95 | 0.44 | 0.61 | 0.30 | 1.00 | 0.40 | 0.78 | 0.13 | 1.00 | 0.32 |

TABLE VII
EVALUATION RESULTS FOR SENTIMENT ANALYSIS TOOLS.

| tool | # correct | positive precision | positive recall | negative precision | negative recall |
|---|---|---|---|---|---|
| *SentiStrength* | 48 | 0.73 | 0.23 | 0.35 | 0.34 |
| *SentiStrength-SE* | 11 | 0.78 | 0.05 | 0.44 | 0.09 |
| *NLTK* | 30 | 0.83 | 0.17 | 0.67 | 0.14 |
| *SentiCR* | 8 | 0.00 | 0.00 | 0.80 | 0.18 |
| *Senti4SD* | 21 | 0.72 | 0.09 | 0.57 | 0.18 |
| *Stanford CoreNLP* | 63 | 1.00 | 0.15 | 0.29 | 0.93 |
| *Pattern matching* | 166 | 0.92 | 0.99 | 0.94 | 0.73 |

TABLE VIII
PRECISION FOR POME AND OPINER IN ASPECT & SENTIMENT PREDICTION.

| predicted aspect | aspect prediction | | sentiment prediction | |
|---|---|---|---|---|
| | POME | **Opiner** | POME | **Opiner** |
| Community | 1.00 | 0.00 | 1.00 | 0.50 |
| Compatibility | 0.36 | 0.33 | 0.45 | 0.33 |
| Documentation | 0.75 | 0.54 | 0.69 | 0.54 |
| Functional | 0.75 | 0.16 | 0.76 | 0.16 |
| Performance | 0.79 | 0.58 | 0.68 | 0.43 |
| Reliability | 0.57 | 0.46 | 0.57 | 0.42 |
| Usability | 0.67 | 0.24 | 0.78 | 0.41 |
| **Overall** | **0.72** | **0.28** | **0.73** | **0.38** |

As an example, when a sentence clearly reports that the library provides some useful features (*"the Commons Configuration project from Apache will do the job; it will allow you to write and read Properties files"*) the pattern-based approach is able to correctly identify it as a positive opinion, while all the sentiment analysis tools label it as neutral. The same happens for the sentence *"as already stated above there is a compatibility issue with mockito-all"*, in which the pattern-based approach is able to recognize the presence of a negative feeling from the compatibility point of view, while the sentiment analysis tools classify the sentence as neutral. Note that this is a limitation of these tools in the specific context in which we are using them. However, this does not mean that they do not work when assessing the sentiment polarity in other contexts (*e.g.,* users' happiness on Stack Overflow).

Given the above results, in POME we rely on the pattern-matching approach to identify sentiment polarity, rather than using existing sentiment analysis tools.

**RQ$_3$** ***How does POME perform compared to Opiner, a state-of-the-art tool for mining opinions from Stack Overflow?*** To answer RQ$_3$, we compare the results of both aspect detection and sentiment analysis achieved by POME and Opiner on the sentences they extracted from Stack Overflow.

Results shown in Table VIII indicate that POME achieves an overall better precision. That is, when POME identifies an aspect from a discussion, the chance of it being correct is higher than that identified by Opiner (0.72 *vs* 0.28).

According to Fisher's exact test, the difference is statistically significant ($p$-value$< 0.001$) with an OR=6.6, *i.e.,* POME has 6.6 times more chances of providing a correct aspect classification than Opiner. The same trend holds for each aspect except "compatibility", where both Opiner and POME exhibit low performance. One example of misclassification by POME in this category is "*it did not work for me with my spring-boot version*", classified by POME as compatibility-related (due to the pattern "did not work [...] [proper noun] version"). The study participants labeled the sentence as not reporting any opinion, probably because it is not clear whether the problem experienced by the user is an actual compatibility issue (as opposed, *e.g.,* to a misuse of the API by the user).

POME significantly outperforms Opiner when identifying opinions related to "usability" and "functional" aspects, with the Fisher's exact test indicating that differences are statistically significant (adjusted $p$-value$< 0.001$). In other cases differences are not statistically significant on single categories because of the small number of samples. However, the ORs are always in favor of POME, ranging from 1.1 for "compatibility" to 16.0 for "functional". We can conclude that POME performs better than Opiner in aspect identification. Since for most aspects POME can achieve a precision greater than 0.6, we can say that the opinions mined by POME are generally reliable, considering that a random assignment of aspect would result in a precision of 1/8 (0.125).

We qualitatively discuss some examples related to functional-related sentences, in which POME obtains a 0.75 precision as compared to the 0.16 achieved by Opiner. Examples of sentences correctly classified in this aspect by POME are "*you can do most of this config using application.properties if you are using spring-boot*", and "*the Guava library has an Ordering.greatestOf method that returns the greatest K elements from an Iterable [...]*". Concerning the misclassifications related to the functional aspects, one of the POME's patterns causing false positives is "[with|use] [library] [pronoun] [helping verb] [verb]" (see [22] for an explanation of this pattern) that matches, for example, the sentence "*if you are* ***using mongo-java-driver*** *then* ***you can have*** *a look at this SO answer*". This pattern was responsible for 7 out of the 33 false positives in the functional aspect. However, it also helped in identifying 8 true positives, thus posing the usual recall *vs* precision dilemma. As for Opiner, its precision in identifying opinions about functional aspects is quite low.

Misclassifications here include "*I am working on a jersey web service*" or "*an important architectural difference is that GWT-RPC operates at a more functional level*". Probably, this is due to the features (words) used by the machine learner to classify the aspects. Indeed, "service" and "functional" are likely to be keywords characterizing feature-related sentences.

When comparing the results of sentiment prediction, POME almost doubles the precision of Opiner (0.73 *vs* 0.38), and performs better in all categories. Fisher's exact test indicates that the observed differences are, again, statistically significant for "functional" and "usability" (adjusted $p$-value$< 0.001$ in both cases). In other cases the test did not report significant differences, again because of the limited number of samples. The ORs are always in favor of POME, ranging from 1.6 of *"compatibility"* to 16.7 of *"functional"*. On the overall dataset, we have a statistically significant difference ($p$-value$< 0.001$) and an OR=4.3, *i.e.,*POME has four times more chances of Opiner in indicating the correct sentiment polarity. Also for what concerns the sentiment prediction the strongest difference between the two approaches is observed in the functional-related sentences. Since we already discussed this category for the aspect identification, we focus our qualitative analysis on the compatibility-related sentences, the ones exhibiting the smaller difference in sentiment prediction precision among the two approaches (0.45 *vs* 0.33). Here, the POME's misclassifications are mostly due to the wrong handling of negations, often caused by misspelling/typing issues. For example, POME misclassifies as positive the sentiment of the sentence "*the problem is that FrameLayout.LayoutParams constructor doesn`t support another FrameLayout as a parameter until the api 19.*" due to the use of the backtick instead of an apostrophe, which caused the negation handling failure. Other examples are typos like "*cann't*" instead of "*can't*". Integrating a spell checker could solve the problem, although it must cope with having source code words not being correct English words.

Concerning Opiner, the main problem is represented by sentences considered by the participants as do not actually reporting an opinion and, thus, being neutral in terms of sentiment while classified as positive/negative by the tool. This is the case for "*BTW, I'm working with Spring MVC*", classified as a positive compatibility sentence by Opiner and as non-opinionated by participants.

Despite the better results achieved by POME, **Opiner identifies a higher number of opinions for these APIs (4 times higher than that identified by** POME**), thus very likely exhibiting a higher recall**. POME has been designed to favor precision over recall, and in $RQ_3$ we are only focusing on the precision of the mined opinions, since assessing the recall would require the analysis of the entire Stack Overflow. The precision reported in $RQ_3$ is not as high as for the other RQs. This might depend on the specific dataset and/or on whom performed the labeling. The datasets used in the previous RQs have been created by the authors, having a deeper knowledge of the problem. Also, they discussed cases where there was a disagreement, while this did not happen for $RQ_3$ participants. Although instructions were given for evaluators of $RQ_3$, the annotation task remains highly subjective. In spite of these concerns, POME advances the current state-of-the-art in aspect and sentiment identification. Also, the difficulty annotators had in their task highlights once more that grasping API opinions from Stack Overflow sentences is not an easy task, and therefore recommenders such as POME and Opiner are valuable.

## VI. Threats to Validity

*Construct validity*. This affects the creation of the labeled dataset used in $RQ_1$ and $RQ_2$. The threat has been mitigated by having multiple evaluators classifying aspects and sentiments.As for the slightly modified approach by Treude and Robillard [42], we manually validated all the sentences extracted with the fourth regular expressions introduced by us in order to discriminate between sentences referring to APIs. Among the 10,481 sentences extracted by our *Fine-grained linker*, 360 have been identified using the fourth rule in Table I. One of the authors manually analyzed all of them, classifying $74\%$ of the sentences correctly linked to the API [22].

*Internal validity*. It is possible that a different calibration of the machine learners produce better results. Therefore, results reported in Table VI and Table V represent a lower-bound for the different configurations of POME.

*Conclusion validity*. Where needed we supported our claims through appropriate statistical procedures. As for the aspect-specific comparison, it is possible that Type-II errors occurred (failed to reject hypothesis due to limited sample), however we showed how the differences were statistically significant on the overall dataset.

*External validity*. While we have validated POME, and compared it with Opiner, on unseen data, it is possible that a different dataset would exhibit different results. Also, another dataset could exhibit different distributions of the identified aspects, and, possibly, further aspects we did not consider. However, this still makes the approach applicable, possibly by augmenting the set of identified patterns. POME is suitable for popular APIs, due to the large availability of opinions to mine. However, this applies to any recommender based on (historical) data mining.

## VII. Conclusion

Mining opinions on APIs helps developers take better decisions during software development. We proposed a pattern-based approach named POME which achieves high accuracy in identifying opinion aspects. POME also outperforms a state-of-the-art tool (Opiner [45]). POME aids developers to quickly gain an understandings of the overall quality, pros, and cons of APIs. As opinions are embedded in many other kinds of sources, our future work is given.

## Acknowledgments

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects PROBE (SNF Project No. 172799) and CCQR (SNF Project No. 175513), and CHOOSE for sponsoring our trip to the conference.

## REFERENCES


[1] "Apache Maven Central Repository," http://central.maven.org/maven2/maven/, last access 24.08.2018.

[2] "Spacy," https://spacy.io.

[3] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: A customized sentiment analysis tool for code review interactions," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017, pp. 106–111.

[4] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empir Software Eng*, 2017.

[5] F. Calefato, F. Lanubile, and N. Novielli, "Emotxt: A toolkit for emotion recognition from text," in *Proceedings of ACII 2017 (7th International Conference on Affective Computing and Intelligent Interaction)*, 2017.

[6] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proceedings of ICSE 2013 (35th International Conference on Software Engineering)*. IEEE press, 2013, pp. 582–591.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, pp. 321–357, 2002.

[8] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proceedings of RSSE 2012 (3rd International Workshop on Recommendation Systems for Software Engineering)*. IEEE Press, 2012, pp. 85–89.

[9] F. L. de la Mora and S. Nadi, "Which library should I use?: a metric-based comparison of software libraries," in *40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER)*, 2018, pp. 37–40.

[10] R. Fisher, "On the interpretation of $\chi^2$ from contingency tables, and the calculation of p," *Journal of the Royal Statistical Society*, vol. 85, no. 1, pp. 87–92, 1922.

[11] M. Goldman and R. Miller, "Codetrail: Connecting source code and web resources," *Journal of Visual Languages & Computing*, pp. 223–235, 2009.

[12] M. Goul, O. Marjanovic, S. Baxley, and K. Vizecky, "Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering," in *Proceedings of HICSS 2012 (45th Hawaii International Conference on System Sciences)*, 2012, pp. 4168–4177.

[13] E. Guzman, R. Alkadhi, and N. Seyff, "An exploratory study of twitter messages about software applications," *Requirements Engineering*, vol. 22, no. 3, pp. 387–412, 2017.

[14] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: an empirical study," in *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 352–355.

[15] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Proceedings of RE 2014 (22nd International Requirements Engineering Conference)*. IEEE, 2014, pp. 153–162.

[16] S. Holm, "A simple sequentially rejective Bonferroni test procedure," *Scandinavian Journal on Statistics*, vol. 6, pp. 65–70, 1979.

[17] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of KDD 2004 (10th ACM SIGKDD international conference on Knowledge discovery and data mining)*, 2004, pp. 168–177.

[18] C. J. Hutto and E. Gilbert, in *Proceedings of ICWSM 2014 (8th International AAAI Conference on Weblogs and Social Media*.

[19] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *Proceedings of MSR 2017 (14th International Conference on Mining Software Repositories)*. IEEE Press, 2017, pp. 203–214.

[20] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Software Engineering*, pp. 1–42, 2017.

[21] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes, "Automatically locating relevant programming help online," in *Proceedings of VL/HCC 2012 (2012 IEEE Symposium on Visual Languages and Human-Centric Computing)*, 2012, pp. 127–134.

[22] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Replication package." https://pome-repo.github.io/.

[23] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for so ware engineering: How far can we go?" p. preprint, 2018.

[24] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Proceedings of SSE 2015 (7th International Workshop on Social Software Engineering)*, ser. SSE 2015, 2015, pp. 33–40.

[25] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive?: empirical study of affectiveness vs. issue fixing time," in *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*. IEEE Press, 2015, pp. 303–313.

[26] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, pp. 1–135, 2008.

[27] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *Proceedings of ICSME 2015 (31st International Conference on Software Maintenance and Evolution)*, ser. ICSME 2015, 2015, pp. 281–290.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[29] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming Prompter," in *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM Press, 2014, pp. 102–111.

[30] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, and M. Lanza, "Supporting software developers with a holistic recommender system," in *Proceedings of ICSE 2017 (39th International Conference on Software Engineering)*, 2017, pp. 94–105.

[31] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of ICSE 2013 (35th International Conference on Software Engineering)*. IEEE Press, 2013, pp. 832–841.

[32] M. P. Robillard and Y. B. Chhetri, "Recommending reference API documentation," *Empirical Software Engineering*, pp. 1–29, 2014.

[33] N. Sawadsky and G. Murphy, "Fishtail: from task context to source code examples," in *Proceedings of TOPI 2011 (1st Workshop on Developing Tools as Plug-ins)*. ACM, 2011, pp. 48–51.

[34] V. Sinha, A. Lazar, and B. Sharif, "Analyzing developer sentiment in commit logs," in *Proceedings of MSR 2016 (13th International Conference on Mining Software Repositories)*. ACM, 2016, pp. 520–523.

[35] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *In Proceedings of EMNLP 2013 (2013 Conference on Empirical Methods in Natural Language Processing)*. Citeseer, 2013.

[36] R. Souza and B. Silva, "Sentiment analysis of travis ci builds," in *Proceedings of MSR 2017 (14th International Conference on Mining Software Repositories)*. IEEE Press, 2017, pp. 459–462.

[37] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding api components and examples," in *Proceedings of VL/HCC 2006 (2006 IEEE Symposium on Visual Languages and Human-Centric Computing)*, 2006, pp. 195–202.

[38] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of ICSE 2014 (36th International Conference on Software Engineering)*, 2014, pp. 643–652.

[39] W. Takuya and H. Masuhara, "A spontaneous code recommendation tool based on associative search," in *Proceedings of SUITE 2011 (3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation)*. ACM, 2011, pp. 17–20.

[40] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the Association for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[41] P. Tourani, Y. Jiang, and B. Adams, "Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem," in *Proceedings of CASCON 2014 (24th Annual International Conference on Computer Science and Software Engineering)*. IBM Corp., 2014, pp. 34–44.

[42] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from stack overflow," in *Proceedings of ICSE 2016 (38th International Conference on Software Engineering)*, 2016, pp. 392–403.

[43] G. Uddin and F. Khomh, "The opiner tool," goo.gl/2EnL78.

[44] ——, "Automatic summarization of api reviews," in *Proceedings of ASE 2017 (32nd IEEE/ACM International Conference on Automated Software Engineering)*. IEEE, 2017, pp. 159–170.

[45] ——, "Opiner: an opinion search and summarization engine for apis," in *Proceedings of ASE 2017 (32nd IEEE/ACM International Conference on Automated Software Engineering)*, 2017, pp. 978–983.

[46] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Proceedings of ASE 2013 28th IEEE/ACM International Conference on Automated Software Engineering)*, 2013, pp. 562–567.

[47] Y. Zhang and D. Hou, "Extracting problematic API features from forum discussions," in *Proceedings of ICPC 2013 (21st International Conference on Program Comprehension)*, 2013, pp. 141–151.