Characterizing Leveraged Stack Overflow Posts

Salvatore Geremia[1], Gabriele Bavota[2], Rocco Oliveto[1], Michele Lanza[2], Massimiliano Di Penta[3]
[1]*University of Molise,* [2]*Universita' della Svizzera italiana (USI),* [3]*University of Sannio*
s.geremia@studenti.unimol.it, {gabriele.bavota, michele.lanza}@usi.ch,
rocco.oliveto@unimol.it, dipenta@unisannio.it

***Abstract*—Stack Overflow is the most popular question and answer website on computer programming with more than 2.5M users, 16M questions, and a new answer posted, on average, every five seconds. This wide availability of data led researchers to develop techniques to mine Stack Overflow posts. The aim is to find and recommend posts with information useful to developers. However, and not surprisingly, not every Stack Overflow post is useful from a developer's perspective. We empirically investigate what the characteristics of "useful" Stack Overflow posts are. The underlying assumption of our study is that posts that were used (referenced in the source code) in the past by developers are likely to be useful. We refer to these posts as *leveraged* posts. We study the characteristics of *leveraged* posts as opposed to the *non-leveraged* ones, focusing on community aspects (*e.g.,* the reputation of the user who authored the post), the quality of the included code snippets (*e.g.,* complexity), and the quality of the post's textual content (*e.g.,* readability). Then, we use these features to build a prediction model to automatically identify posts that are likely to be *leveraged* by developers. Results of the study indicate that post meta-data (*e.g.,* the number of comments received by the answer) is particularly useful to predict whether it has been leveraged or not, whereas code readability appears to be less useful. A classifier can classify leveraged posts with a precision of 65% and recall of 49% and non-leveraged ones with a precision of 95% and recall of 97%. This opens the road towards an automatic identification of "high-quality content" in Stack Overflow.**

I. Introduction

Software developers often need to acquire new pieces of knowledge to deal with the ever-increasing complexity of modern software systems. Thus, developers are engaged with a continuous information-seeking process performed by interacting with teammates, by reading different forms of documentation, and by consulting online resources such as question and answer (Q&A) websites.

Q&A websites have become a prominent point of reference for software developers [1], also due to the vast availability of the information they provide. Stack Overflow (SO), the most popular Q&A website related to computer programming, counts, at the date of writing, over 16 Million questions and 25 Million answers[1]. Such a vast amount of data represents a precious source of information that can be automatically mined to provide support for software engineering tasks.

Indeed, many researchers proposed recommender systems built on top of the information mined from SO. These include, for example, techniques recommending SO discussions relevant for a given task at hand [2], [3], providing support for the automatic documentation of source code [4]–[6], or recommendation of code elements [5], [7], [8].

One of the main challenges for these recommender systems is the identification of SO posts of *high-quality* and that could be useful for developers. Researchers have dealt with this challenge by integrating heuristics aimed at discarding low-quality content. For example, AutoComment [4], a tool using information from SO posts to automatically document source code, only uses information items from posts that have been positively judged by the SO community. Although heuristics can be in many cases appropriate to provide accurate recommendations, they do not consider the extent to which the recommended posts reflect properties of posts that, in the past, have been considered useful by developers. Thus, the following questions remain unanswered:

> *What is a useful SO post? Is it an answer that received many up-votes, or coming from a well-reputed user? Do the properties of snippets contained in the post matter?*

Given that many SO answers have inspired solutions in open-source projects [9], [10], we conjecture that answers useful in the past for somebody are likely to be useful in the future. Thus, we empirically investigate which are the characteristics of SO answers that have been previously "leveraged" by developers, and compare them with that of "non-leveraged" answers. We assume an answer to have been leveraged if it has been mentioned (*i.e.,* linked) at least once in the source code of an open source project hosted on GitHub. Such a link might indicate the willingness of a developer to (i) indicate the reuse of the answer's code snippets, (ii) document the rationale of an implementation choice, or (iii) simply refer to the answer as an interesting source of information. **Note that, while for a linked answer it is safe to assume that it has been leveraged, a non-linked answer might have been leveraged without being explicitly linked. For example, the code snippet contained in the answer might have been copied without putting a proper attribution in the source code about the snippet's provenance [11]. We discuss how this issue impacts our findings in Section III-A**.

We use data from SOTorrent [12], an open database containing the official SO data dump and including references from GitHub files to SO posts. We analyze the last available release (2018-02-16) containing 42 Million SO posts and 6 Million post references from GitHub. We filter out from this set the subset of data referring to SO answers including at least one

[1]https://stackoverflow.blog/2018/09/27/stack-overflow-is-10/

Java code snippets. This was done to (i) have a homogeneous set of posts for which it is possible to compute community aspects (*e.g.,* the reputation of the user who authored the post), quality of the included code snippets (*e.g.,* complexity), and quality of the post's textual content (*e.g.,* readability); (ii) check whether the snippets of the answers classified as non-leveraged (*i.e.,* not explicitly linked) have been reused in GitHub projects without a proper attribution, thus threatening our classification of the answer as "non-leveraged" (details in Section II). Through this filtering, we obtained 19,342 posts, 3,214 of which *leveraged* at least once in projects hosted in GitHub, and 16,128 that were not leveraged.

Using this dataset, we first statistically compare the distribution of 22 factors related to community aspects, quality of the code snippets, and quality of the posts' text, between *leveraged* and *non-leveraged* posts. Then, we use such factors as independent variables to build classifiers able to predict whether a post is likely to be leveraged or not (dependent variable). We experiment with four different classifiers (*i.e.,* Bayesian Network, J48, Logistic Regression, and Random Forest) in several configurations.

Results show that the Random Forest is the classifier obtaining the best classification accuracy, with AUC=0.856 and Matthews Correlation Coefficient (MCC)=0.528. More specifically, the classifier achieves a precision of 65% and recall of 49% for *leveraged* posts, and a precision of 95% and recall of 97% for *non-leveraged* posts. While our model is still far from providing a highly precise classification, the achieved results pave the way to more research on the automatic identification of "high-quality content" in SO.

II. Study Design

The *goal* of this study is to investigate the characteristics of SO answers that have been previously *leveraged* by developers, as opposed to other answers (*non-leveraged*) for which there is no evidence of their usefulness. The *perspective* is of researchers interested to understand what makes a good SO answer, and possibly to exploit this information to better recommend SO posts. The study *context* consists of 19,342 answers posted on SO (3,214 leveraged and 16,128 non-leveraged), all related to the Java programming language, and always containing a source code snippet.

In our study we assume a SO answer to have been leveraged if it has been linked at least once in the code of a GitHub project. However, a non-linked answer might have been leveraged without being explicitly linked [11]. For this reason, we formulate the following preliminary research question:

> **RQ_0:** *To what extent are code snippets from "non-leveraged" answers used in GitHub projects?*

We answer RQ_0 by selecting a sample of SO answers classified as *non-leveraged* in our dataset (through the process detailed later on in the study design), with the goal of verifying how many of their code snippets have been reused in Java GitHub projects without an explicit reference. This will give an idea of how reliable is our classification of SO answers as leveraged and non-leveraged. As shown in Section III-A, only a small percentage of *non-leveraged* is misclassified, meaning that its code snippet has been used in a GitHub project without a proper reference.

After this preliminary analysis, we investigate whether the value distributions for various kinds of answer's features, characterizing both its content and its author, change between *leveraged* and *non-leveraged* answers. Therefore, we ask our first research question:

> **RQ_1:** *Which are the characteristics of SO answers that have been leveraged by developers?*

We focus on three kinds of properties that can be objectively measured in SO answers and, thus, we compared them in *leveraged* and *non-leveraged* answers: (i) community aspects including, the reputation of the user who posted the answer, whether the answer has been marked as the "accepted answer" by the user who asked the question, etc.; (ii) the quality of the code snippet included in the answer, assessed using state-of-the-art quality metrics that can be measured on a (possibly incomplete) code snippet (*e.g.,* cyclomatic complexity, readability); (iii) the quality of the answer's textual content, mainly assessed through metrics capturing its readability. The complete list of features is described in detail below and reported in Table I.

We use the knowledge acquired answering $\mathbf{RQ}_1$, and in particular we use the factors studied in $\mathbf{RQ}_1$ as independent variables to devise an approach based on a machine learning technique to predict whether a given SO answer will be leveraged or not (dependent variable). Therefore, we pose our second research question:

> $\mathbf{RQ}_2$: *Which is the accuracy of a recommender system in identifying posts that are likely to be leveraged by developers?*

In the context of $\mathbf{RQ}_2$, we also investigate how the prediction accuracy is influenced by (i) the choice of the machine learning algorithm, (ii) the balancing of the training set, and (iii) the amount and temporal recency of the posts used to build the training set. Finally, we analyze the importance of each independent variable for prediction purposes.

A. Data Collection

As the first step to answer our research questions, we collected SO answers that have been *leveraged* or *non-leveraged*. We use SOTorrent, a dataset provided by Baltes *et al.* [12] and containing the official SO data dump "augmented" with information about links going from GitHub files to SO posts. Thanks to SOTorrent, it is possible to know which SO posts have been linked in open source projects hosted on GitHub. In our study, we decided to focus only on SO answers since we measure on these posts a number of characteristics that are only available for answers (*e.g.,* whether an answer is the one marked as *accepted* by the user who posted the question). Also, we decided to only consider answers containing at least one Java code snippet. Again, this is done since (i) among the post characteristics we study, we also consider code quality aspects of the included snippets, *i.e.,* complexity, readability,

TABLE I
FACTORS CONSIDERED IN OUR STUDY

Factor	Description
COMMUNITY FACTORS	
Is Accepted	1 if the answer is accepted, 0 otherwise
Answer Score	Answer upvotes minus answer downvotes
Comment Count	The number of answer's comments
Creation Date	The date when the answer was created
User Reputation	A score summarizing the reputation of a user on SO (see [13])
User Up Votes	The number of *upvotes* received by a user
User Down Votes	The number of *downvotes* received by a user
CODE QUALITY FACTORS	
Snippet LOC	The lines of code of the answer's code snippet(s)
Snippet Complexity	The cyclomatic complexity [14] of answer's code snippet(s)
Snippet Readability	The readability of the answer's code snippet(s) [15]
TEXT READABILITY FACTORS	
# Words	The number of words in the answer text
# Sentences	The number of sentences in the answer text
# Characters	The number of characters in the answer text
# Syllables	The number of syllables in the answer text
# Complex Words	The number of words composed of at least 3 syllables
ARI [16]	$4.71 \cdot \frac{\#Characters}{\#Words} + 0.5 \cdot \frac{\#Words}{\#Sentences} - 21.43$
SMOG [17]	$1.043 \cdot \sqrt{\#ComplexWords \cdot \frac{30}{\#Sentences}} + 3.1291$
SMOG Index [17]	$\sqrt{\#ComplexWords \cdot \frac{30}{\#Sentences}} + 3$
Flesch Kincaid [18]	$0.39 \cdot \left(\frac{\#Words}{\#Sentences}\right) + 11.8 \cdot \left(\frac{\#Syllables}{\#Words}\right) - 15.59$
Flesch Reading Easy [19]	$206.835 - 1.015 \cdot \left(\frac{\#Words}{\#Sentences}\right) - 84.6 \cdot \left(\frac{\#Syllables}{\#Words}\right)$
Coleman Liau [20]	$\left(5.89 \cdot (\frac{\#Characters}{\#Words})\right) - \left(30 \cdot (\frac{\#Sentences}{\#Words})\right) - 15.8$
Gunning Fog [21]	$0.4 \cdot \left[(\frac{\#Words}{\#Sentences}) + 100 \cdot (\frac{\#ComplexWords}{\#Words})\right]$

and size, and (ii) the code snippet will be used in Section IV to verify whether the code from non-leveraged answers have been reused (without an explicit link to the answer), thus threatening our the classification of posts as non-leveraged.

We used Google BigQuery [22] to select from the table `Posts` all answers that are *leveraged* (*i.e.,* linked in at least one Java file) and that contain at least one code snippet:

```
SELECT * FROM [sotorrent-org:2018_12_09.Posts]
WHERE Id IN (
  SELECT PostId
  FROM [sotorrent-org:2018_12_09.PostReferenceGH]
  WHERE FileExt = ".java")
AND PostTypeId = 2
AND Body LIKE "\%<pre\%<code>\%</code>\%</pre>\%"
```

This query returned 3,437 *leveraged* answers. Then, we select from the table `Posts` the *non-leveraged* answers containing at least one Java code snippet:

```
SELECT * FROM [sotorrent-org:2018_12_09.Posts]
WHERE Id NOT IN (
  SELECT PostId
  FROM [sotorrent-org:2018_12_09.PostReferenceGH])
AND PostTypeId = 2
AND Body LIKE
  "\%<pre class=\"lang-java\%<code>\%</code>\%</pre>\%"
```

This query returned 18,592 *non-leveraged* answers.

One important difference must be noticed between the two used queries. In the query selecting the *non-leveraged* answers, we made sure that the included code snippet was manually set to be a Java snippet by the SO user posting the answer.

This can be seen from the `class="lang-java"` property used in the query. This property is optional, and can be set by the SO user to obtain a better formatting of the posted code snippet. By querying the SOTorrent dataset, we noticed that only a minority of answers, even among those reporting a Java snippet, used the `class="lang-java"`.

Since the number of *non-leveraged* answers is much greater than the number of the *leveraged* answers, we decided to use this filtering when collecting the *non-leveraged*, since, in any case, the number of returned answers was high enough to run our study (18,592). This was not the case for *leveraged* answers, with only a few dozens answers using the `class="lang-java"` property. For this reason, we applied a "weaker" filter to identify *leveraged* answers, simply ensuring that the answer was linked in at least one Java file. However, through manual analysis, we noticed that not all linked answers include a Java snippet. For example, in some cases developers linked in a Java file a Stack Overflow answer containing a snippet written in C, just to indicate that they reimplemented in Java the algorithm in the posted answer.

Since, as detailed later, our study infrastructure is tailored for Java, we applied the following cleaning process on the 3,437 *leveraged* answers. First, we considered as relevant for our study, all the *leveraged* answers satisfying at least one of three conditions:

1) It was posted in response to a question explicitly marked with the "java" tag;
2) It was posted in response to a question containing the word "java" in the title;
3) It contained a code snippet explicitly marked as a Java snippet, as previously discussed for the *non-leveraged* answers.

This process resulted in 1,904 *leveraged* answers classified as relevant for our study. Then, we excluded the 17 answers in which the included snippet was explicitly marked as non-Java (*e.g.,* `class="lang-php"`). Finally, the first author manually analyzed the remaining 1,516 answers (3,437-1,904-17=1,516), selecting for inclusion in our study those that included a Java snippet. In the end, we considered as valid 3,228 *leveraged* answers out of the 3,437 initially obtained by querying SOTorrent.

Finally, we excluded all answers posted after Jan, 1 2018 (note that SOTorrent reported, in the analyzed database, posts up to Dec 2018). This choice was dictated by the following observation: A very recent post could be *non-leveraged* simply because no one "had enough time" to leverage it, and not due to its characteristics. For this reason, we excluded almost one year of data from our study, to factor out, at least in part, the "time" confounding factor. This left us with 19,342 posts, of which 3,214 *leveraged* and 16,128 *non-leveraged*.

For each of the selected answers, we extracted the following set of characteristics, or factors, listed in Table I:

- **Community Factors.** This category includes characteristics of the answers that are related to their presence on a Q&A website. We consider factors acting as proxies for the "quality" of the answer (*i.e., Is Accepted* and *Answer Score*); factors assessing the reputation of the user who posted the answer (*i.e., User Reputation*, *User Up Votes*, and *User Down Votes*); and factors representing metadata of the answer, meaning its *Creation Date* and the number of comments it received (*i.e., Comment Count*).

- **Code Quality Factors.** We only consider in our study answers including at least one Java code snippet. For a given answer, we take the set of contained code snippets, and we concatenate them together as a single snippet. Then, we measure its size, in terms of Lines Of Code (*Snippet LOC* in Table I), its complexity, assessed with McCabe Cyclomatic Complexity [14], and its readability, assessed with the metric proposed by Buse and Weimer [15]. This metric combines a set of low-level code features (*e.g.,* identifiers length, number of loops, etc.) and has been shown to be 80% effective in predicting developers' readability judgments. We used the implementation of such a metric provided by the original authors[2].
- **Text Readability Factors.** We use several state-of-the-art text metrics to assess the readability of the text contained in the answer. These metrics are built on top of some basic information about the text to evaluate, such as the number of words and sentences composing it, the number of used complex words (*i.e.,* those composed of at least three syllables), etc. Starting from this information, different text readability metrics have been proposed. For example, the SMOG readability formula [17] estimates the years of education a reader needs to understand the given text. All text readability metrics used in our study have been computed using an open-source API available on GitHub[3].

While the extracted data is sufficient to answer RQ_1 and RQ_2, the analysis performed does not contemplate the extent to which answers classified as non-leveraged have still been used, without proper reference, in GitHub projects, thus invalidating their classification as "non-leveraged". To deal with this issue, in RQ_0 we selected a sample of 500 answers classified as *non-leveraged* and verified whether their code snippets have been used in GitHub projects. Note that out of $\simeq 16k$ non-leveraged snippets, a sample of 500 ensures a confidence interval of $\pm 4.3\%$ for a confidence level of 95%. Since searching in the whole GitHub is not doable in a reasonable amount of time, we applied the following process to reduce the "search space" (*i.e.,* the files on GitHub where to search for the snippets of interest). We used the StORMed [23] island parser to extract from each of the 500 snippets the data types they use. Then, we excluded the snippets only using primitive types (*e.g.,* `int`) and/or the data types already defined in Java (*e.g.,* `String`). The excluded snippets were replaced with others randomly selected, to meet our goal of sampling 500 snippets for this analysis. A manual analysis of all extracted types was performed to remove noise identified through the island parser. For example, we found snippets using `YourType` or `foo` as data type. Again, these were excluded and replaced with other snippets.

Once completed the collection of the 500 snippets with the related types, we used the GitHub search APIs [24] to search for source code files including, in their text, the name of at least one of the types used by the selected snippets. We only searched in Java files (`language : java`). Since the GitHub APIs only report a maximum of 1,000 results per request (in our case, 1,000 files containing a given type), for each type t we sent several requests searching for Java files containing t and having a size in bytes included in a specific range r. In particular, we started with $r = [0, \ldots, 2500]$ (*i.e.,* all files with a size between 0 and 2.5 kB), storing all found files in a given folder. Then, we increased r at steps of 2500 Bytes (*i.e.,* the second search looked for files containing t and having a size between 2.5 and 5 kB), until we reached the maximum size supported by the GitHub APIs, in which only files smaller than 384 kB are searchable. This process provided us with a total of 230k files containing at least one of the searched types and took approximately 20 days.

Finally, given S one of the 500 snippets, we used the Simian clone detector [25] to identify type-2 clones between S and the set of files downloaded from GitHub using the same type(s) present in S. We chose Simian due to the fact that it can easily be run on non-compilable code. We set five as the minimum number of lines to be matched in order to classify a snippet as cloned in a Java file.

B. Analysis Methodology

We answer $\mathbf{RQ_0}$ by reporting the percentage of analyzed *non-leveraged* answers, for which we found a clone of their code snippet in GitHub project (*i.e.,* the percentage of misclassified *non-leveraged* answers).

We address $\mathbf{RQ_1}$ by statistically comparing the value distributions of the factors described in Table I between *leveraged* and *non-leveraged* posts. Especially, we use the Wilcoxon Rank Sum Test [26] assuming a significance level $\alpha = 5\%$, and the Cliff's delta (d) effect size [27].

We split the data into ten buckets based on the creation date of the answers. In particular, the first bucket contains the 10% oldest answers, while the tenth groups the 10% newest ones. Then, we show how the differences vary when considering datasets having a different size and different recency. We start by only considering the first bucket (oldest 10%). Then, the first two (oldest 20%), and so on up to 90% at steps of 10%.

These datasets (*i.e.,* first 10%, first 20%, etc.) are also the training sets used in RQ_2 where we test the machine learning models on unseen data.

Thus, when performing statistics to address RQ_1, we do not consider the last bucket (most recent 10%). Since the analysis involves multiple comparisons, we adjust p-values using the Benjamini-Hochberg procedure [28], which ranks p-values, keeping the largest p-value unaltered, multiplying the second largest by the number of p-values divided by the rank (*i.e.,* two), and treating the remaining ones similarly to the second.

One of the limits of our study is that there may be other factors, beyond those captured by the considered features, that determined whether or not a post has been leveraged. To mitigate this threat, we identified posts having both leveraged and non-leveraged answers (104 in total in our dataset). For such posts, we performed a paired analysis *within-post* of

[2]Available at http://tinyurl.com/kzw43n6

[3]https://github.com/ipeirotis/ReadabilityMetrics

the feature distribution, this time using a paired test, *i.e.,* the Wilcoxon Rank Sign Test [26], as well as using the paired Cliff's delta effect size [27].

To answer $\mathbf{RQ}_2$, we build classifiers based on machine learning techniques, using the factors in Table I as independent variables (*i.e.,* features) and the categorical variable *leveraged/non-leveraged* as dependent variable. Before applying the machine learners, we identify groups of factors that correlate and, for each group, we only keep the one that better correlates with the dependent variable. In detail, we use the *R varclus* function of the *Hmisc* package, producing a hierarchical clustering of features based on their correlation, in turn, computed with a specified correlation measure (Spearman's ρ rank correlation). Then, we identify clusters by cutting the tree at a given level of ρ^2 that we set at $\rho^2 = 0.64$, which corresponds to a strong correlation [29] (*i.e.,* $\rho = 0.8$). As a result of this analysis, we removed the following features:

- User upvotes (correlates with user reputation);
- # Syllables, # Characters, # Complex Words (all correlate with Words); and
- ARI, SMOG, SMOG Index, and Gunning Fog (all correlate with Flesch-Kincaid).

Fig. 1 reports the clustering dendrogram of features obtained with the *R varclus* function (the y-axis represents the ρ^2).

We used the ten data buckets previously created to train/test the machine learning algorithms. We start using the first bucket as a training set for the models, and the remaining nine as a test set. This process is iterated nine times, increasing each time the amount of data (*i.e.,* number of buckets) used for training by 10%. In any case, we make sure that the answers used for training are older than the ones used for testing. For example, in the second iteration, the first two buckets (*i.e.,* 20% oldest answers) are used for training and the remaining 80% of data for testing. In the last iteration, 90% of data is used for training and 10% for testing.

Such a process was preferred to a 10-fold cross-validation since it avoids the use of "data from the future" when predicting the instances in the test set. Indeed, without using the constraints that answers in the training set must be older than the ones in the test set, there is the risk of using, for example, answers from 2018 to predict whether an answer from 2016 will be leveraged or not. This is clearly not representative of a real usage scenario for the predictor; indeed, when predicting whether a new answer will be leveraged, we can only use data from the past to train the classifier. Also, using this experimental design allows to study the impact on the accuracy of the classifier of (i) the amount of used training data, and (ii) the recency of the data used for training.

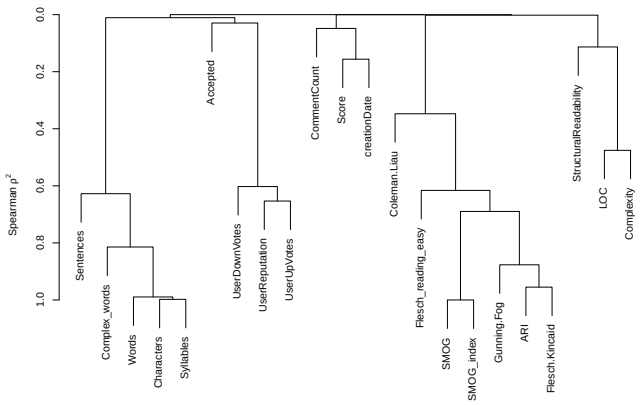

Fig. 1. Feature correlation dendrogram (the y-axis shows Spearman's ρ^2)

We experimented four different machine learning techniques implemented in Weka [30], namely Decision Trees (J48), Bayesian classifiers, Random Forests, and Logistic Regression. We used such techniques with their default configuration.

In our dataset we have many more *non-leveraged* than *leveraged* answers. To take into account such a strong unbalancing, we experimented each machine learning technique when (i) not balancing the training sets; (ii) balancing the training sets by under-sampling the majority class by means of the Weka implementation of the *SpreadSubSample* filter; and (iii) balancing the training sets by generating artificial instances of the minority class using the Weka *SMOTE* filter.

We report Precision, Recall, Accuracy, Area Under the Receiver Operating Characteristics curve (AUC), and Matthews Correlation Coefficient (MCC) [31]. Precision and Recall are reported for both prediction categories: *leveraged* (Precision_l and Recall_l) and *non-leveraged* (Precision_{nl} and Recall_{nl}). We use MCC, since it is a measure used in machine learning assessing the quality of a two-class classifier especially when the classes are unbalanced. It ranges between -1 and 1 (0 means that the approach performs like a random classifier) and it is defined as:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(FN+TN)(FP+TN)(TP+FN)}}$$

The MCC can be interpreted as follows: $MCC < 0.2$ indicates a low correlation, $0.2 \leq MCC < 0.4$ a fair, $0.4 \leq MCC < 0.6$ a moderate, $0.6 \leq MCC < 0.8$ a strong, and MCC ≥ 0.8 a very strong correlation [29].

We also report information about the importance of the considered factors using the Mean Decrease Impurity (MDI) [32], which measures the importance of variables on an ensemble of randomized trees.

C. Replication Package

The data used in our study is publicly available [33]. In particular, we provide (i) the subject answers together with the factors (see Table I) we measured on each of them; (ii) the created buckets, together with the training (both the balanced and the unbalanced) and test sets as `arff` files to be used in *WEKA*; and (iii) the detailed results for the accuracy of each machine learning algorithm.

III. Results

In this section we present and discuss results aimed at addressing the research questions formulated in Section II.

A. To what extent are code snippets from "non-leveraged" answers used in GitHub projects?

We found that, out of the 500 snippets considered as *non-leveraged*, only 30 (6%) have at least one detected clone in the considered GitHub files. Thus, while we acknowledge a certain level of noise in our analysis (*i.e.,* misclassification of leveraged snippets as non-leveraged), we believe that the findings reported in the following are unlikely to be substantially influenced by such a noise.

B. Which are the characteristics of SO answers that have been leveraged by developers?

Table II reports Cliff's delta (*d*) effect size, together with its interpretation (*i.e.,* L=Large, M=Medium, S=Small, N=Negligible) for the studied factors when using buckets of different size and recency, as explained in Section II. A positive effect size value indicates that the value for a given factor (*e.g.,* the *Comment Count*) is higher in the group of leveraged SO answers, while the opposite holds for negative values. Empty cells in Table II represent combinations of factors/datasets for which the obtained *p*-value is not statistically significant.

By looking at Table II, the first thing that leaps to the eyes is that results are consistent among the different buckets, with very minor variations. For example, the *Answer Score* factor always exhibits a large, positive effect size, suggesting that leveraged SO answers usually have higher score values as compared to non-leveraged ones. Based on the available data, this factor seems to be the most prominent one in characterizing leveraged answers, followed by *Comment Count*.

Among the community factors, it is surprising to see that the *Accepted Answer* does not play a major role here. However, it is worth noticing that not all SO questions have an accepted answer. Indeed, as also extensively discussed in the Stack Exchange community[4], not all users take the time to mark as accepted one of the answers they got in response to a question they posted on the website.

Most of the user-related factors (*i.e.,* factors characterizing the user who post the answer) exhibit negligible and not statistically significant differences between leveraged and non-leveraged answers. The only exception here is for the *User Down Votes*, that exhibit a small effect size with the increase in the size of the dataset. It is important to note the negative sign of the effect size, indicating that the number of downvotes is higher in non-leveraged answers.

For what concerns the quality of the code snippet embedded in the answer, the code readability does not seem to play a major role, while snippets that are larger (*Snippet LOC*) and more complex (*Snippet Complexity*), tend to be leveraged more, even though the effect size is small. While this result might look surprising, it indicates that developers are more likely to leverage non-trivial code rather than very simple small snippets. In other words, when developers look for information on SO and link a specific post in their source code, this code is likely to be more complex than that in non-leveraged posts.

The readability of the text contained in the answer (see factors from *# Words* to *Gunning Fog* in Table II) only exhibits negligible differences between the two compared sets of posts, possibly due to the fact that these metrics were not conceived to assess the readability of technical documents, such as the one in SO discussions.

Finally, the post *Creation Date* exhibits a statistically significant difference and medium effect size. The negative values here indicate something quite expected: Oldest posts are more likely to have been leveraged by developers. This finding also justifies our temporal buckets-based approach both in RQ_1 and, especially, in the training of the classifier in RQ_2. Indeed, since the "age" of a post plays a role in our data, it is important to analyze how the accuracy of a classifier varies when training it on older or newer data.

While the analysis of Table II provided us with good insights on the characteristics of leveraged and non-leveraged answers, it is important to highlight that it has been performed on SO answers belonging to different discussions. In particular, for a given SO discussion, it is possible that none of the answers we analyzed have been leveraged, while for others it could happen that all the considered answers have been linked in some open source project.

Despite this does not introduce any noise in our data, it does not help in controlling for possible confounding factors. Indeed, it might be possible that some answers belonging to a SO discussion are not reused because their topic has a very narrow interest, rather than because of the factors we considered, *e.g.,* because it received few comments or because it came from a user with a low reputation.

For these reasons, we replicated the previous analysis when only considering the 104 SO discussions in our dataset having at least one leveraged and one non-leveraged answer (*i.e.,* paired analysis). We consider the dataset composed by the 90% of data, to avoid having a too-small number of discussions to consider for this analysis. Table III reports the achieved results.

On this (much smaller) dataset, most of our main findings are confirmed, meaning the role played by *Answer Score*, *Comment Count*, *User Down Votes*, *Creation Date*. Instead, we did not observe any significant difference for factors related to the quality of the code snippet. However, this might be due to the small size of this dataset. The main difference is that the User-related factors also play a significant role, with a large effect size for *User Reputation*. Also, when focusing only on the answers to a specific question, it seems that developers tend to leverage more accepted answers.

Despite the slightly different results obtained in the unpaired and in the paired scenario, both analyses showed the presence of statistically significant differences in the distribution of some of the considered factors between leveraged and non-leveraged answers. This suggests the possibility of automatically discriminate between these two answers' sets, as we investigate in RQ_2.

[4]https://meta.stackexchange.com/questions/119197/problem-with-users-not-accepting-answers

TABLE II
RQ$_1$: EFFECT SIZE FOR STATISTICALLY SIGNIFICANT DIFFERENCES (N=NEGLIGIBLE, S=SMALL, M=MEDIUM, L=LARGE)

Feature	% Dataset Size								
	10%	20%	30%	40%	50%	60%	70%	80%	90%
Is Accepted									
Answer Score	+.72 (L)	+.79 (L)	+.82 (L)	+.82 (L)	+.81 (L)	+.82 (L)	+.82 (L)	+.82 (L)	+.82 (L)
Comment Count	+.52 (L)	+.54 (L)	+.54 (L)	+.53 (L)	+.53 (L)	+.52 (L)	+.51 (L)	+.50 (L)	+.50 (L)
User Reputation		+.06 (N)	+.04 (N)					-.03 (N)	
User Up Votes			-.05 (N)	-.10 (N)	-.11 (N)	-.10 (N)	-.09 (N)	-.10 (N)	-.09 (N)
User Down Votes		-.07 (N)	-.09 (N)	-.14 (N)	-.16 (S)	-.16 (S)	-.15 (S)	-.17 (S)	-.16 (S)
Snippet LOC		+.07 (N)	+.09 (N)	+.10 (N)	+.12 (N)	+.14 (N)	+.16 (S)	+.16 (S)	+.16 (S)
Snippet Complexity	+.15 (S)	+.18 (S)	+.20 (S)	+.21 (S)	+.22 (S)	+.23 (S)	+.24 (S)	+.24 (S)	+.24 (S)
Snippet Readability							+.03 (N)	+.04 (N)	+.04 (N)
# Words	-.07 (N)	-.05 (N)	-.05 (N)	-.06 (N)	-.04 (N)	-.03 (N)	-.04 (N)	-.04 (N)	-.05 (N)
# Sentences				-.04 (N)					
# Characters	-.07 (N)	-.05 (N)	-.05 (N)	-.06 (N)	-.04 (N)	-.03 (N)	-.04 (N)	-.04 (N)	-.05 (N)
# Syllables	-.07 (N)	-.05 (N)	-.05 (N)	-.06 (N)	-.03 (N)	-.03 (N)	-.03 (N)	-.04 (N)	-.04 (N)
# Complex Words	-.07 (N)	-.05 (N)	-.04 (N)	-.04 (N)					-.03 (N)
ARI	-.07 (N)	-.06 (N)	-.04 (N)	-.04 (N)	-.05 (N)	-.05 (N)	-.05 (N)	-.05 (N)	-.05 (N)
SMOG	-.06 (N)	-.05 (N)				-.03 (N)	-.03 (N)	-.03 (N)	-.03 (N)
SMOG Index	-.06 (N)	-.05 (N)				-.03 (N)	-.03 (N)	-.03 (N)	-.03 (N)
Flesch Kincaid	-.07 (N)	-.05 (N)			-.04 (N)	-.04 (N)	-.04 (N)	-.04 (N)	-.04 (N)
Flesch Reading Easy									
Coleman Liau									
Gunning Fog	-.06 (N)	-.07 (N)	-.04 (N)	-.04 (N)	-.05 (N)	-.05 (N)	-.05 (N)	-.05 (N)	-.05 (N)
Creation Date	-.39 (M)	-.45 (M)	-.43 (M)	-.40 (M)	-.39 (M)	-.36 (M)	-.37 (M)	-.40 (M)	-.44 (M)

TABLE III
RQ$_1$: PAIRED ANALYSIS OF FEATURES' DISTRIBUTION ON 104 DISCUSSIONS HAVING BOTH LEVERAGED AND NON-LEVERAGED ANSWERS

Feature	p-value	d	Magnitude
Is Accepted	<**0.01**	0.36	medium
Answer Score	<**0.01**	0.73	large
Comment Count	<**0.01**	0.76	large
User Reputation	<**0.01**	0.49	large
User Up Votes	**0.01**	0.26	small
User Down Votes	<**0.01**	0.30	small
Snippet LOC	0.44	-0.05	negligible
Snippet Complexity	0.39	0.10	negligible
Snippet Readability	0.14	0.12	negligible
# Words	0.07	0.10	negligible
# Sentences	0.07	0.14	negligible
# Characters	0.07	0.09	negligible
# Syllables	0.08	0.08	negligible
# Complex words	0.14	0.06	negligible
ARI	0.27	-0.10	negligible
SMOG	0.14	-0.14	negligible
SMOG Index	0.14	-0.14	negligible
Flesch Kincaid	0.14	-0.13	negligible
Flesch Reading Easy	0.14	0.15	small
Coleman Liau	0.39	-0.07	negligible
Gunning Fog	0.14	-0.12	negligible
Creation Date	<**0.01**	-0.65	large

RQ$_1$ summary: post-related features such as *Answer Score*, *Comment Count* and post *Creation Date* exhibit significant and large/medium differences between leveraged and non-leveraged posts. Snippets in leveraged posts are longer and more complex than in non-leveraged posts. Finally, readability metrics play a negligible role.

C. Which is the accuracy of a recommender system in identifying posts that are likely to be leveraged by developers?

Table IV reports the accuracy of the machine learning classifiers in the different configurations we experimented. Table IV reports the overall results achieved across the nine buckets used for the training. This means that for each run (*i.e.,* the first run refers to the usage of 10% for training, 90% for testing; the second 20%-80%; etc. up to 90%-10%), we counted the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Then, we count the overall number of TP as the sum of the TP identified in all nine runs and apply the same procedure also for TN, FP, FN. Based on such numbers, we computed the statistics reported in Table IV.

The Random Forest consistently ensures slightly better performance than the other algorithms, independently from the adopted balancing strategy. The balancing of the training set does not help the Random Forest in obtaining a better classification. Indeed, its performance is quite stable, with the balancing (both under and oversampling) helping the recall of the *leveraged* class while penalizing its precision. In general, the performance achieved by the top configuration, while being far from those of a perfect model, are satisfactory.

The AUC of 0.856 indicates that the model is much better than a random classifier (AUC=0.5), and the MCC reports a moderate correlation (0.528). Particularly relevant for the purpose of our study is the precision achieved by the classifier when identifying *leveraged* answers (Precision$_l$=0.654). This means that 65% of the answers identified as *leveraged* by the classifier have been linked in at least one open-source project on GitHub.

Once identified the best classifier/configuration, we further dig into factors that affect its performance, starting from the amount and recency of training data. As explained in Section II, we excluded from our study answers posted after Jan 1, 2018, to avoid recent posts. Starting from this dataset (from now on referred to as D^1_{18}), we study the impact of the recency of used data by creating other four datasets, obtained excluding from D^1_{18} the most recent answers at steps of six months. Thus, the first dataset includes only answers posted until Jul 1, 2017 (D^7_{17}), the second until Jan 1, 2017 (D^1_{17}), and so on, until D^1_{16}. This analysis aims at verifying our conjecture that older data are more "reliable" to build our prediction model since "old" answers that have never been leveraged are unlikely to be leveraged in the future, while recent *non-leveraged* answers might not being exploited yet

TABLE IV
RQ$_2$: PREDICTION ACCURACY OF THE EXPERIMENTED CLASSIFIERS IN DIFFERENT CONFIGURATIONS. CONFIGURATIONS ARE REPORTED IN DESCENDING ORDER BY MCC (MATTHEWS CORRELATION COEFFICIENT)

Machine Learner	SMOTE	Under sampling	Recall$_l$	Precision$_l$	Recall$_{nl}$	Precision$_{nl}$	Accuracy	AUC	MCC
RandomForest	✗	✗	0.490	0.654	0.973	0.949	0.928	0.856	0.528
RandomForest	✓	✗	0.521	0.615	0.966	0.952	0.925	0.866	0.526
RandomForest	✗	✓	0.591	0.537	0.948	0.958	0.914	0.869	0.516
J48	✓	✗	0.529	0.552	0.956	0.952	0.916	0.790	0.494
J48	✗	✗	0.510	0.553	0.958	0.950	0.916	0.743	0.485
Bayesian	✗	✗	0.441	0.583	0.968	0.944	0.918	0.848	0.464
Bayesian	✗	✓	0.510	0.508	0.949	0.950	0.908	0.845	0.458
J48	✗	✓	0.592	0.437	0.922	0.956	0.891	0.746	0.449
Bayesian	✓	✗	0.315	0.655	0.983	0.933	0.921	0.833	0.419
Logistic	✗	✗	0.150	0.764	0.995	0.919	0.916	0.777	0.315
Logistic	✗	✓	0.234	0.524	0.978	0.925	0.909	0.762	0.309
Logistic	✓	✗	0.241	0.461	0.971	0.926	0.903	0.762	0.287

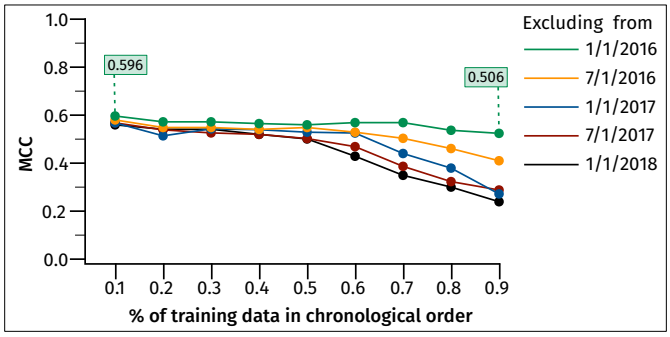

Fig. 2. RQ$_2$: MCC achieved by the Random Forest with different datasets and percentage of training data

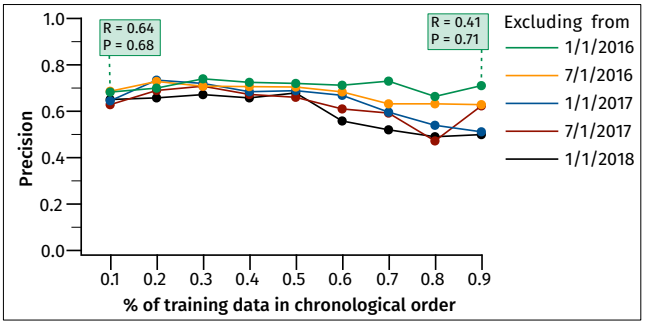

Fig. 3. RQ$_2$: Precision$_l$ achieved by the Random Forest with different datasets and percentage of training data

due to their recency. To also study the impact of the training data on the accuracy of the prediction model, we split also these four datasets into ten buckets, as already described for the original D_{18}^1 dataset (*i.e.,* the first 10% contains the oldest answers).

Fig. 2 and 3 report the MCC and the Precision$_l$, respectively, obtained by the Random Forest (i) on the five different datasets, containing SO answers characterized by different recency (see the five lines having different colors), and (ii) when using different buckets for training (*i.e.,* oldest 10%, 20%, etc., see x-axis).

As conjectured, older data ensures better prediction accuracy. Indeed, the green line, indicating the D_{16}^1 dataset, is on the top of both graphs, thus showing a higher MCC/Precision$_l$ achieved by the Random Forest in D_{16}^1 as compared to newer datasets. Second, the amount of training data does not strongly influence the model's accuracy. The MCC goes down with the increase of the training set size. However, this result can be explained by the interaction between the "post recency" and the "training set size" factors. Indeed, due to our experimental design, when we increase the training set size, we add more recent answers to it (since the buckets are chronologically ordered). Thus, if we move from 10% to 90% of training data, we are adding in the training set answers that are much more recent than those contained in the first bucket (10%). As previously shown, the Random Forest works better on older data. Thus, the possible boost in accuracy given by the increase of the training dataset size is counterbalanced by the increased recency of the training answers.

TABLE V
RQ$_2$: MEAN DECREASE IMPURITY (MDI) OF FEATURES USED TO PREDICT POST REFERENCE

Feature	MDI
Is Accepted	0.502 ± 0.020
Comment Count	0.416 ± 0.006
Answer Score	0.396 ± 0.003
User Reputation	0.372 ± 0.004
LOC	0.356 ± 0.003
User Down Votes	0.351 ± 0.002
Snippet Complexity	0.323 ± 0.009
Snippet Readability	0.322 ± 0.005
# Words	0.304 ± 0.006
# Sentences	0.296 ± 0.005
Flesch Kincaid	0.284 ± 0.005
Flesch Reading Easy	0.268 ± 0.003
Coleman Liau	0.260 ± 0.007
Creation Date	0.243 ± 0.006

Finally, Table V reports the values of Mean Decrease in Impurity (MDI) of features used to predict *leveraged* answers. The results refer to the D_{18}^1 dataset: We computed the MDI values when using each of the nine training sets (from 10% to 90% of data) and, then, computed the average and the 95% confidence interval for each feature.

Table V confirms the major role played by features related to *community factors* (*i.e., Is Accepted*, *Comment Count*, *Answer*

Score, *User Reputation*, and *User Down Votes*). Following, are the features characterizing the answer's code snippet (*i.e., LOC*, *Snippet Complexity*, and *Readability*), while less relevant for the prediction are factors related to the readability of the answer's text.

RQ_2 summary: The Random Forest is the classifier ensuring the best accuracy, with an AUC of 0.86 and an MCC of 0.53. The classification accuracy is strongly influenced by the recency of the data, indicating that the categorization of older answers as *leveraged* and *non-leveraged* is more reliable than that of more recent answers. The *community factors* are considered as the most important features to discriminate between *leveraged* and *non-leveraged* answers.

IV. Threats to Validity

Construct validity. The most crucial threat is the way we determine whether a Stack Overflow post has been leveraged. As explained in Section II, we rely on the dataset created by Baltes *et al.* [12], which use post hyperlinks in the source code to determine that a post has been leveraged. Possibly, a better approach, which we plan to use in our future work, would be to complement Baltes *et al.*'s heuristic with clone detection [9] which, however, would be very expensive from a computational point of view due to the need for searching each of the ∼18k *non-leveraged* snippets in the whole GitHub. As explained in Section II, at least we give an estimation of the percentage of *non-leveraged* snippets that are likely to be misclassified (4%).

Internal validity. We used default settings for the machine learning algorithms, therefore it is possible that a better calibration could improve their results. Nevertheless, this means that the achieved results represent a lower bound of the classifications' performance. Although we consider features characterizing SO posts from different perspectives, it is entirely possible that the developers' criteria for choosing whether to leverage a post or not are based on factors we do not capture. We have mitigated this threat by performing RQ_1, a paired, *within-post*, analysis of the considered features for 104 posts having both leveraged and non-leveraged answers.

External validity. We are aware that our work is (i) limited to Java-related posts (due to the need for using Java metrics extraction, in particular, readability metrics), and (ii) it relies on observable evidence of post leverages as reported by Baltes *et al.* [12]. Further work is needed to replicate the study on posts containing snippets written in other languages.

V. Related work

We discuss the related literature about (i) reuse of code from the Internet, (ii) empirical models built on top of SO data, and (iii) development of recommender systems based on SO.

A. Reusing Code From the Internet

Xia *et al.* [34] showed that one of the common activities performed by software developers while searching on the Web is to look for code snippets to reuse. Based on their results, it is not surprising that many researchers studied the reuse of code snippets across the Internet.

Sojer and Henkel [35] focused on the legal and economic risks of code reuse from the Internet. They surveyed 869 professional developers to investigate whether the reuse of code snippets from the Internet is a common practice in commercial software. Furthermore, the analysis shows a growth in the importance of code reuse from the Internet in recent years. For such reasons, work aimed at better characterizing SO posts worthwhile of being reused — like the one proposed in our work — is highly desirable.

Baltes and Diehl [10] presented a large-scale empirical study investigating the ratio of unattributed usages of SO code snippets on GitHub. In particular, they analyzed the *copy-paste* and *ad-hoc* usage of code snippets into public Java GitHub projects. The results of their study show that half of the developers copying SO snippets do not put proper attribution for the reused code, resulting in three-quarter of reused code snippets that are not correctly attributed on GitHub projects. Attribution and licensing issues are out of the scope of our work, but, for example, licensing constraints are certainly a factor that can constrain code reuse [36].

Yang *et al.* [37] analyzed more than 3 million SO code snippets across four programming languages: C#, Java, JavaScript, and Python, to study the usability of snippets contained in *accepted answers*. They found that less than 4% of Java snippets are parsable and only 1% are compilable. The situation is substantially different for Python, for which they found 72% of code snippets to be parsable and 26% to also be runnable. Unlike our study, Yang *et al.* were interested in understanding whether code snippets *can be easily reused*, while we focus on factors influencing the reuse of SO posts on GitHub.

Yang *et al.* [9] analyzed 1.9 Million Python code snippets posted on SO and more than 900k Python projects on Github to investigate if SO code snippets are used in real projects. They performed three types of analysis: a perfect match between the SO snippet and the GitHub code, differences due to syntactic elements, and partial clones. They found 1,566 copy-paste blocks, 9,044 blocks which vary for spaces, tabs etc., and more than 405k GitHub blocks that are partial clones of SO snippets. Rather than using clone detection, we use references inside source code (from Baltes *et al.* dataset [12]) to capture reuse, also because as explained in the introduction we are interested to capture reuse in a broad sense, not limited to snippet copy-paste. However, in future, the SOTorrent dataset could be combined with clone detection results.

Other works analyzed the reuse of SO code snippets in Android apps [38]–[40], looking at their impact on the app's code quality in terms of reliability [38] and security [39], [40].

A specific study on snippets licensing and attribution was conducted by Le An *et al.* [11]. By analyzing 399 Android apps, they found 266 snippets potentially reused from SO, 1,226 SO posts containing examples from the apps, and a total of 1,279 licensing violations. Considering licensing is complementary to our purpose: once we determine whether a post has been leveraged, it might be necessary to determine

whether one could legally reuse its code or not.

B. Prediction Tasks on Stack Overflow

Related to our work are also approaches that try to predict/classify properties on SO items (*e.g.,* questions, answers).

Regarding the prediction of questions quality, Correa and Sureka [41] proposed a model to detect questions that are likely to be deleted.

They used 47 features divided into four categories: question content (*e.g.,* number of URLs, code snippet length), syntactic style (*e.g.,* number of words in title/body), user profile (*e.g.,* number of previous questions/answers), and community-generated (*e.g.,* average question score, average number of accepted answers). To train the model, they selected 235k deleted and non-deleted questions (470k in total). Results show that the model is able to correctly classify both types of questions with a 66% accuracy.

Xia *et al.* [42] proposed an approach to predict deleted questions which combines two independent classifiers, one based on textual features and the other built on the 47 features used by Correa and Sureka [41]. Results show that combining multiple features can help to better discriminate deleted from non-deleted questions.

Ponzanelli *et al.* [43] used machine learning and genetic algorithms to identify poor-quality questions at their creation time. They used three families of metrics as independent variables to assess the questions' quality: SO metrics (*i.e.,* length of the question, the textual similarity between title and body, number of tags), readability metrics, and popularity metrics (*i.e.,* badged answer and question count, and badges-tags coverage). The "question quality" dependent variable is considered to be *good*, if the question is not closed, not deleted, with an accepted answer, and with a score between 1 and 6, *very good*, if it is *good*, but with a greater score, *bad*, if it is not closed, not deleted, and with a score below 0, and *very bad*, if it is closed or deleted. Results show that the popularity of the author is more important than textual features to determine the quality of a new question. In our work, we also consider author- and post-related features and, indeed, our results indicate that some of them are good predictors of reuse.

There have also been different works dealing with the prediction of relevant tags for SO questions. Xia *et al.* [44] proposed TagCombine, a framework able to analyze objects in software information sites and provide tags recommendations. Saini and Tripathi [45] and Wang *et al.* [46] also proposed approaches to predict SO question tags. We do not consider how tags could influence reuse, but this is certainly a direction for future work.

In summary, while several studies investigated the characteristics of SO posts, to the best of our knowledge, our study is the first to investigate the possibility to *predict* SO posts that are likely to be leveraged in open source projects.

C. Stack Overflow in Recommender Systems

Among the various sources available on the Web, Q&A Websites and in particular SO, have been exploited by many recommender systems for software developers to identify pieces of information relevant for a given task at hand. Examples of these systems include (but are not limited to): techniques to identify code elements contained in SO answers [47]; the automatic recommendation of SO discussions for a given task run by the developer in the IDE [2], [3]; and support for the automatic documentation of source code using information mined from SO [4]–[6].

Due to lack of space, we do not discuss work on SO recommenders in detail. However, it is important to highlight that our study can disclose empirical evidence about the possibility of automatically identify SO posts that are "worth reusing". Such information can be of paramount importance for the development of better recommender systems able to select relevant posts *as humans would do*.

VI. Conclusion

In this paper we studied the characteristics of Stack Overflow (SO) answers that have been *leveraged* by developers, who explicitly linked such answers in their source code. Different characteristics of the *leveraged* answers, including user-related, post-related, and snippet-related characteristics, have been compared to those of *non-leveraged* posts. Results of the comparison highlighted, through statistical analysis, significant differences between *leveraged* and *non-leveraged* answers. In particular, the differences were large for what concerns post-related and community-related factors, such as the quality of the answer as perceived by the SO users (*e.g.,* answer score), and the engagement it created on the platform (*e.g.,* comment count).

Starting from this result, we experimented with machine learning algorithms, assessing their ability to automatically discriminate between *leveraged* and *non-leveraged* answers. We obtained encouraging results (AUC=0.856 and MCC=0.528). This points to the possibility of integrating, in recommender systems, "filtering" mechanisms able to identify "high-quality" content in the sea of data available on Q&A websites. This present work is thus only a first step in that area, paving the way for several research directions.

A more comprehensive study can be run by considering as leveraged posts not only those explicitly linked in source code files of open source projects, but also those from which code snippets have been reused [9]. This can help in enlarging the set of leveraged posts, better studying their characteristics and, possibly, improving the performance of the prediction models.

Related to this point is the possibility to factor into our models additional predictor variables. For example, whether the code snippet included in the answer is easy to reuse (*e.g.,* can be parsed [37]). Indeed, our models use a limited set of predictor variables that can be easily expanded.

Finally, the long term goal is the integration of the prediction models able to discriminate high-quality content into one of the recommender systems exploiting SO (*e.g.,* [3]) to verify, through user studies, whether they contribute to recommend more relevant information items.

REFERENCES

[1] M. Umarji, S. Sim, and C. Lopes, "Archetypal internet-scale source code searching," in *Proceedings of OSS 2008 (The 4^{th} International Conference on Open Source Systems)*, 2008, pp. 257–263.

[2] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proceedings of RSSE 2012*. IEEE Press, 2012, pp. 85–89.

[3] 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 the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. ACM, pp. 102–111.

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

[5] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. ACM, 2016, pp. 392–403.

[6] E. Aghajani, G. Bavota, M. Linares-Vasquez, and M. Lanza, "Automated documentation of android apps," *Transactions on Software Engineering (TSE)*, 2019.

[7] A. Zagalsky, O. Barzilay, and A. Yehudai, "Example overflow: Using social media for code recommendation," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, 2012, pp. 38–42.

[8] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 513–522.

[9] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack overflow in github: any snippets there?" in *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, 2017, pp. 280–290.

[10] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *arXiv preprint arXiv:1802.02938*, 2018.

[11] L. An, O. Mlouki, F. Khomh, and G. Antoniol, "Stack overflow: A code laundering platform?" in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, 2017, pp. 283–293.

[12] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts," *arXiv preprint arXiv:1803.07311*, 2018.

[13] "Stack overflow reputation. https://stackoverflow.com/help/whats-reputation."

[14] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[15] R. P. L. Buse and W. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2010.

[16] R. Senter and E. A. Smith, "Automated readability index," CINCINNATI UNIV OH, Tech. Rep., 1967.

[17] G. H. Mc Laughlin, "Smog grading-a new readability formula," *Journal of reading*, vol. 12, no. 8, pp. 639–646, 1969.

[18] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," 1975.

[19] R. Flesch, "A new readability yardstick." *Journal of applied psychology*, vol. 32, no. 3, p. 221, 1948.

[20] M. Coleman and T. L. Liau, "A computer readability formula designed for machine scoring." *Journal of Applied Psychology*, vol. 60, no. 2, p. 283, 1975.

[21] R. Gunning, *The technique of clear writing*. McGraw-Hill, 1968. [Online]. Available: https://books.google.ch/books?id=vJZpAAAAMAAJ

[22] "Google bigquery. https://cloud.google.com/bigquery/."

[23] L. Ponzanelli, A. Mocci, and M. Lanza, "Stormed: Stack overflow ready made data," in *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*. ACM Press, 2015, pp. 474–477.

[24] "Github search apis. https://developer.github.com/v3/search/."

[25] S. Harris, "Simian - similarity analyser. https://www.harukizaemon.com/simian/."

[26] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

[27] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[28] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society, Series B*, vol. 57, no. 1, p. 125–133, 1995.

[29] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Lawrence Earlbaum Associates, 1988.

[30] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[31] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 1975.

[32] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 431–439.

[33] R. O. M. L. M. D. Salvatore Geremia, Gabriele Bavota.

[34] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Software Engineering*, vol. 22, no. 6, pp. 3149–3185, 2017.

[35] M. Sojer and J. Henkel, "License risks from ad hoc reuse of code from the internet," *Communications of the ACM*, vol. 54, no. 12, pp. 74–81, 2011.

[36] C. Vendome, D. M. Germán, M. Di Penta, G. Bavota, M. L. Vásquez, and D. Poshyvanyk, "To distribute or not to distribute?: why licensing bugs matter," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 268–279.

[37] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: an analysis of stack overflow code snippets," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 391–402.

[38] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from stackoverflow: An exploratory study on android apps," *Information and Software Technology*, vol. 88, pp. 148–158, 2017.

[39] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 289–305.

[40] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 121–136.

[41] D. Correa and A. Sureka, "Chaff from the wheat: characterization and modeling of deleted questions on stack overflow," in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 631–642.

[42] X. Xia, D. Lo, D. Correa, A. Sureka, and E. Shihab, "It takes two to tango: Deleted stack overflow question prediction with text and meta features," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 73–82.

[43] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza, "Understanding and classifying the quality of technical forum questions," in *Quality Software (QSIC), 2014 14th International Conference on*. IEEE, 2014, pp. 343–352.

[44] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 287–296.

[45] T. Saini and S. Tripathi, "Predicting tags for stack overflow questions using different classifiers," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*. IEEE, 2018, pp. 1–5.

[46] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, Apr 2018.

[47] P. Rigby and M. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of ICSE 2013*, 2013, pp. 832–841.