Automatic Identification of Code Smell Discussions on Stack Overflow: A Preliminary Investigation

Sergei Shcherban, Peng Liang*
School of Computer Science
Wuhan University
Wuhan, China
liangp@whu.edu.cn

Amjed Tahir
School of Fundamental Sciences
Massey University
Palmerston North, New Zealand
a.tahir@massey.ac.nz

Xueying Li
School of Computer Science
Wuhan University
Wuhan, China
xueyingli@whu.edu.cn

ABSTRACT

Background: Code smells indicate potential design or implementation problems that may have a negative impact on programs. Similar to other software artefacts, developers use Stack Overflow (SO) to ask questions about code smells. However, given the high number of questions asked on the platform, and the limitations of the default tagging system, it takes significant effort to extract knowledge about code smells by means of manual approaches. **Aim:** We utilized supervised machine learning techniques to automatically identify *code-smell* discussions from SO posts. **Method:** We conducted an experiment using a manually labeled dataset that contains 3000 *code-smell* and 3000 *non-code-smell* posts to evaluate the performance of different classifiers when automatically identifying code smell discussions. **Results:** Our results show that Logistic Regression (LR) with parameter C=20 (inverse of regularization strength) and Bag of Words (BoW) feature extraction technique achieved the best performance amongst the algorithms we evaluated with a precision of 0.978, a recall of 0.965, and an F1-score of 0.971. **Conclusion:** Our results show that machine learning approach can effectively locate *code-smell* posts even if posts' title and/or tags cannot be of help. The technique can be used to extract code smell discussions from other textual artefacts (e.g., code reviews), and promisingly to extract SO discussions of other topics.

CCS CONCEPTS

• **Software and its engineering → Designing software**; • **Machine learning → Supervised learning by classification**.

KEYWORDS

Code Smell, Discussion, Automatic Classification, Stack Overflow

ACM Reference Format:
Sergei Shcherban, Peng Liang, Amjed Tahir, and Xueying Li. 2020. Automatic Identification of Code Smell Discussions on Stack Overflow: A Preliminary Investigation. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3382494.3422161

*Corresponding author

1 INTRODUCTION AND BACKGROUND

A code smell is a sign that normally corresponds to more rooted problems in the program [12]. A bad smell might not directly correspond to faults, but it may serve as an indicator of a part of the code that is error-prone. The presence of code smells may also indicate that a developer did not follow good design principles and coding best practices, results in a harder to understand and maintain code. Previous empirical studies found that developers generally have a negative perception of code smells [19, 30], whose bad impact on software quality was reported in several studies (e.g., [1, 14]).

Stack Overflow (SO) is probably the most popular Q&A platform and used by millions of developers. As of April 2020, there are over 19 million questions and more than 29 million answers on SO, asked by over 12 million users[1]. There are other similar forums such as Stack Exchange Software Engineering[2] and Code Project[3]; however, SO is the largest in terms of size and the number of users[4], and therefore we select SO for this study. Given its large size, it may take a significant effort to manually search for relevant knowledge on the platform.

Finding and locating related posts and discussions from SO is a challenge due to the large number of questions and the nature of the tagging systems used. To this end, we aim to provide an automatic approach that can help developers and users in locating useful code smell discussions at SO, without depending on existing tags or titles (which both have proven to be ineffective and limited in finding relevant SO posts) [5, 25].

To obtain relevant questions and discussions, developers can use *tags* to search through SO posts. SO depends on users to tag their questions (correctly) by selecting relevant tags to the topic of the question. Moderators can also update and add relevant tags. However, tags have been proven to be ineffective in many cases [5]. There are several problems with using tags as the exclusive approach to decide whether a post is related to a specific topic. A user who publishes a question could be unsure about the title of the most relevant tag for their discussion, which can lead to the use of wrong or irrelevant tags. Another problem with user-defined tags is that users may try to add as many tags as possible (SO allows up to 5 tags) to raise the number of views and probably increase the probability of getting responses quickly [4]. Therefore, while tags

[1]https://stackexchange.com/sites
[2]https://softwareengineering.stackexchange.com/
[3]https://www.codeproject.com/
[4]https://insights.stackoverflow.com/survey/2019

can be useful to find questions related to *code smells*, using tags exclusively may miss relevant questions. Also, the tagging system depends on the user deciding the most relevant tag, leaving tags as an unreliable (and somehow biased) option.

A recent work on code smell discussion on SO [26] and later on other Stack Exchange sites [25] found that many of the code smell related questions did not contain any code smell identifiers (i.e., terms) in the tags or titles. In a recent move, SO removed the "code-smell" tag as the topic was considered to be subjective (which is against SO policy)[5]. This may result in posts that contain relevant code smell knowledge but cannot be located if they use tags that are not within the scope of the search. On the other hand, manual search for posts can be time-consuming. The amount of content from SO would prevent any full reliable manual methods to search through and classify all unlabeled posts [28]. Previous studies have found that categorizing SO questions automatically will make it easier for developers to locate related answers to their questions [21]. **In this work**, we employed a machine learning approach to automatically identify *code-smell* discussions from SO posts. First, we used manually labelled *code-smell* posts obtained from Tahir et al., [26], together with randomly selected *non-code-smell* posts that we collect separately. In order to classify posts automatically, we conducted an experiment using five supervised machine learning algorithms (i.e., Random Forest (RF), k-Nearest-Neighbor (kNN), Support Vector Machine (SVM), Gaussian Process (GP), and Logistic Regression (LR)) and six feature extraction techniques (i.e., BoW, TF-IDF, Word2Vec, Doc2Vec, FastText, and GloVe). We measured the performance of the classification algorithms using a set of metrics (i.e., accuracy, recall, precision, and F1-score). Our results show that LR with parameter C=20 and BoW as a feature extraction technique achieved the best performance with an accuracy of 0.971, a precision of 0.978, a recall of 0.965, and an F1-score of 0.971.

2 RELATED WORK

Over the past few years, SO has been extensively used to investigate topics and trends discussed by software developers. In this section, we discuss previous work related to 1) code smells, and 2) techniques used in mining information from textual software artefacts.

2.1 Code Smell

Most of the previous work on code smells focused on studying the influence of code smells on software quality attributes, such as change- and defect-proneness [14], code readability, and understandability [1]. Other works also studied the impact of various forms of code smells such as architectural smells [13] and test smells [6]. Of those studies, Olbrich et al. [17] found that classes with God and Brain Class smells are more likely to be change and defect-prone, and therefore hard to maintain. Tufano et al. [29] showed that approximately 80% of code smells are never eliminated from a software system, as they tend to stay in the program for long.

2.2 Mining Information from Textual Artefacts

Several analytics techniques have been applied to mine question categories from SO posts in order to provide a better understanding of the knowledge from SO. Treude et al. [28] were among the first to study SO questions' categories. By manually analysing a sample of 385 posts, they found 10 different question categories. Beyer and Pinzger [9] used a card sorting technique to classify 450 Android related SO posts, and found 8 main question types related to Andriod. Based on the manually labeled dataset, they used a kNN algorithm to automatically classify all posts (which achieved a low precision of 0.413). Allamanis et al. [2] used Latent Dirichlet Allocation (LDA), an unsupervised machine learning algorithm, to cluster SO posts into certain categories with 0.98 cosine similarity of question types. We also automatically identified architecture smell discussions from SO using machine learning algorithms [27].

Recent studies have achieved good results when using supervised learning approaches to categorize SO questions (e.g., [7]). Of those studies, Beyer et al. [8] used RF and SVM on a dataset of 1000 manually labelled questions from SO, then classified these questions into 7 categories with an average precision and recall of 0.900. Ahasanuzzaman et al. [3] used Conditional Random Field and several supervised machine learning algorithms to classify API-related questions, and achieved a high precision in classifying relevant questions by using CAPS (LR model). Also, the authors identified the impact of using different sources of information (title, body, answers, Conditional Random Field, and experience features). The results showed that the more information sources are used, the more accurate the classification algorithms work.

3 RESEARCH DESIGN

The goal of this study is to investigate how we can automatically identify *code-smell* discussions from SO. We first used a manually labelled *code-smell* posts dataset that we obtained from the previous work of Tahir et al. [26], and then combined it with a set of *non-code-smell* posts (our data collection is explained in Section 3.1). With this dataset, we investigated two Research Questions (RQs):

RQ1: Which machine learning algorithm works best for identifying *code-smell* discussions from SO posts?

The aim of this RQ is to find the best classification algorithm (in terms of performance) when training classifiers to identify *code-smell* discussions from SO. Different machine learning algorithms can produce different results, depending on the parameters and configurations used. Our choice fell on five machine learning algorithms, including RF, kNN, SVM, GP, and LR, because they are frequently used in binary classification. We conducted five runs using BoW, which is the simplest feature extraction technique, and the five machine learning algorithms with default parameters (see Section 3.2, Phase 4), to train the classification models. We evaluated the performance of each algorithm using accuracy, precision, recall, and F1-score metrics.

RQ2: Which combination of feature extraction techniques and machine learning algorithm works best for identifying *code-smell* discussions from SO posts?

This RQ aims at finding the best configuration of the algorithm's parameters and feature extraction techniques that achieves the best performance. Feature extraction techniques have a significant impact on the results. We plan to experiment with the best machine learning algorithm from RQ1 using different C parameters (0.001, 0.01, 0.1, 1, 10, 20, 30) and feature extraction techniques, including BoW, TF-IDF, Word2Vec, Doc2Vec, FastText, and GloVe.

[5]https://meta.stackoverflow.com/questions/306314/we-are-not-reeking-of-code-smell

Table 1: Examples of code-smell and non-code-smell post

Type	Example
Code-smell post	*"Let s say for example that I m making a tile map editor. We have the editor, which handles the drawing of the tiles, and we have the tile-set which is used to determine what tiles are drawn. The editor needs to depend on the tile-set to know which tiles should be draw, and the tileset needs to depend on the editor to know the dimensions of the tiles to be drawn, as well as other minor details.This creates tightly coupled code. Is this a code smell? If so, how do I resolve it? Do I stuff everything into a large class? Do I use a mediator to communicate between the two classes?"*
Non-code-smell post	*"I have a system that combines the best and worst of Java and PHP. I am trying to migrate a componentthat was once written in PHP into a Java One. Does anyone have some tips for how I can parse a PHP serialized datastructure in Java? By serialized I mean output from php's serialize function."*

3.1 Data Collection

For constructing the dataset, we selected posts that contain *code-smell* discussions provided by Tahir et al. in [26]. The dataset contains over 3000 code smell related discussion mined from SO using a hybrid automatic and manual filtration process. To provide a balanced dataset, since the standard machine learning algorithms yield better prediction performance with balanced dataset [15], we initially scraped a few thousands posts from SO that do not contain the terms "*code-smell*", "*smell*", or "*anti-pattern*" in the posts body, and we then randomly collected 3000 *non-code-smell* posts. One author manually labelled *non-code-smell* posts, and two authors cross-validated the contents of the dataset using a sample size of 341 randomly selected posts (with 95% confidence level and 5% margin of error). We did not find any false positives in the sample. Our dataset has been provided online for replication and reproduction purposes [23]. Table 1 shows examples of what would be considered as *code-smell*[6] and *non-code-smell* post[7].

3.2 Posts Classification Process

The process of classifying code smell discussions is composed of five phases:

Phase 1: Input Data. The input is the dataset which contains 3000 *code-smell* posts and 3000 *non-code-smell* posts from SO and we further split the dataset into training set and testing set.

Phase 2: Preprocess Text. The text is preprocessed to obtain valid textual information (removing useless characters, such as '\r', '\t', '\', and stop words).

Phase 3: Feature Extraction. We applied feature extraction techniques (i.e., BoW to answer RQ1 and BoW, TF-IDF, Word2Vec, Doc2Vec, FastText, GloVe to answer RQ2) to extract features of posts.

Phase 4: Train Classification Algorithms. To answer RQ1, we trained five classification algorithms (RF, kNN, GP, SVM, and LR) with default parameters to classify *code-smell* posts. To answer RQ2, we trained LR (the best algorithm in RQ1) with different parameters[8] *C* (Inverse of regularization strength).

Phase 5: Evaluate Trained Classification Algorithms. We evaluated the performance of each algorithm from **Phase 4** using multiple performance measures.

In **Phase 2** (i.e., text preprocessing), we used NLTK, a toolkit for symbolic and statistical NLP [10], to remove special characters and stop words. The experiment dataset contains a total of 6000 posts from SO. We divided the dataset into *testing* and *training* sets: 33% of posts (1980) as the testing set and 67% of posts (4020) as the training set.

Before applying any machine learning algorithms, it is important to decide which feature extraction techniques to use. Many researchers have applied different techniques such as BoW [22], TF-IDF [24], or more complex techniques such as Word2Vec [18] and Doc2Vec [16] to extract features from a text that can be used as an input for the algorithms. Stanik et al. [24] has found that BoW performs as as high as TF-IDF, while Shah et al. [22] shows that using a simple BoW has performed almost as well as a model with more complex techniques, such as Convolutional Neural Network (CNN); with an F1-scores of 0.66 and 0.65, respectively.

Therefore, and to answer RQ1, we used the BoW feature extraction technique. BoW [31] is a simple technique to represent textual objects for machine learning classification algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the dataset. BoW uses all unique terms in the SO posts as textual features and uses term frequency as the weight of textual features. We used CountVectorizer as a scikit-learn implementation of BoW. When classifying the posts, we selected the whole post body as the input. Specifically, we first created a vocabulary of all the unique words occurring in the dataset. Then using BoW, a *code-smell* discussion *j* is expressed by a vector $X_j = (x_{1,j} \ldots x_{i,j} \ldots x_{n,j})$, in which $x_{i,j}$ denotes the weight of feature *i* calculated by the frequency of term *i* in *code-smell* discussion *j*, and *n* denotes the number of terms in the vocabulary. That is, we used raw weights (i.e., terms frequency) as the word scores.

After getting the best machine learning algorithm from RQ1, we plan to further evaluate which feature extraction technique works best. TF-IDF combines term frequency with inverse document frequency in order to obtain the weight of textual features, which in turn is influenced by the term frequency in the dataset. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently than others. Word2Vec is based on the assumption that words appearing often in the same contexts have similar meanings. Unlike the continuous BoW algorithm, Word2Vec uses Skip-gram (a phrase with a skip), and Negative Sampling is used to make training more efficient: models are provided with words that are not contextual neighbors. Doc2Vec extends

[6]https://stackoverflow.com/questions/33678643
[7]https://stackoverflow.com/questions/98090

[8]parameters set as default in the scikit-learn library

Word2Vec towards documents, and encodes whole documents instead of the lists of ungrouped sentences. GloVe [20] tries to solve the problem of using matching statistics effectively, and minimizes the difference between the product of word vectors and the logarithm of the probability of their co-occurrence by using stochastic gradient descent. FastText [11] learns word representations by taking into account subword information, i.e., incorporating character n-grams into the skipgram model. **To answer RQ2**, we used BoW, TF-IDF, Word2Vec, Doc2Vec, FastText, and GloVe as feature extraction techniques. We then evaluated the best feature extraction techniques with different parameters, such as *n-gram* parameter equals 1, 2, and 3 to determine optimal parameters. As the best machine learning algorithm from RQ1, we used LR with different values of parameter C (Inverse of regularization strength). The standard value of C is 1 (smaller values indicate stronger regularization), and we used the following values: 0.001, 0.01, 0.1, 1, 10, 20, 30.

3.3 Performance Evaluation

We evaluated the performance of each algorithm based on accuracy, precision, recall, and F1-score, which are commonly used in performance evaluation of information retrieval, and we applied 10-fold cross-validation on the dataset. The *code-smell* posts that have been correctly identified by a classifier are considered as True Positive (TP) while the *non-code-smell* posts that have been incorrectly identified are considered as False Positive (FP). The *code-smell* posts incorrectly labelled as *non-code-smell* posts are considered as False Negative (FN). Accuracy is the proportion of accurately predicted set from the whole observations. Precision is the ratio of posts correctly identified as *code-smell* posts to all posts identified as *code-smell* posts. Recall is the fraction of all *code-smell* posts correctly demarcated. F1-score is the harmonic mean of precision and recall. We calculated those metrics using the following equations:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$F1 - score = \frac{2TP}{2TP + FP + FN} \tag{2}$$

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

4 RESULTS AND ANALYSIS

To determine the best configuration for classifying *code-smell* posts, we applied the following two-step approach: In the first step, we evaluated the performance of each combination of machine learning algorithms and BoW. For the second step, we used the best algorithm from the first step with different feature extraction techniques and C parameters. Overall, we experimented with 1 (BoW) × 5 (five classification algorithms)+ (6 feature extraction techniques) × 1 (classification algorithm) + 1 (classification algorithm) × 4 (BoW and TF-IDF with *n-gram* equals 1, 2, 3) × 7 (C parameters) experiment configurations. These experiment configurations were applied on a training set of 4020 posts and a testing set of 1980 posts.

RQ1: Which machine learning algorithm works best for identifying *code-smell* discussions from SO posts?

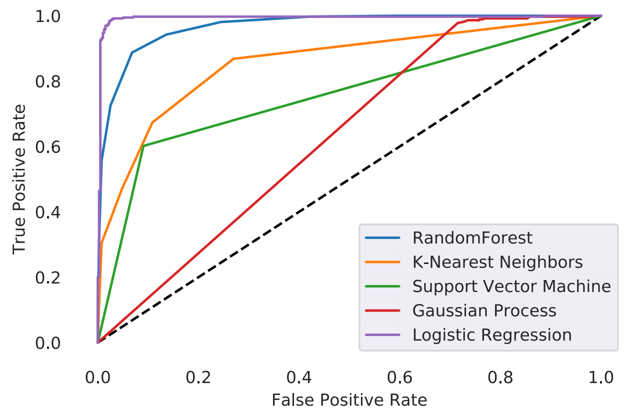

Figure 1: Receiver operating characteristic curve of various algorithms with BoW

To answer RQ1, we used the default parameters for all five algorithms and BoW as a feature extraction technique. Table 2 shows the average accuracy, precision, recall, and F1-score for each algorithm. The best performed algorithm was LR (accuracy = 0.970, precision = 0.978, recall = 0.962, and F1-score = 0.970), while the least performed algorithm was GP (accuracy = 0.689, precision = 0.903, recall = 0.440, and F1-score = 0.592). Figure 1 shows the receiver operating characteristic curve of various algorithms with BoW. The precision-recall curve of various algorithms with BoW is shown in Figure 2. In the five classification algorithms, LR is a more fitting machine learning algorithm in the classifier training step when identifying *code-smell* discussions from SO compared to other classification algorithms.

Table 2: The performance of the classification of posts by various machine learning algorithms with BoW

	Accuracy	Precision	Recall	F1-score
RF	0.885	0.930	0.839	0.882
kNN	0.712	0.927	0.474	0.627
SVM	0.764	0.895	0.611	0.726
GP	0.689	0.903	0.440	0.592
LR	0.970	0.978	0.962	0.970

As we can see from Table 2, algorithms like kNN or GP had a higher FN-ratio compared to the other three algorithm, as they tend to predict many *code-smell* posts as *non-code-smell* posts. In contrast, algorithms such as LR (and to a lesser extend, RF) had a significantly lower numbers of type I and type II errors. This have allowed these algorithms to achieve significantly better results in the identification of *code-smell* posts.

RQ2: Which combination of feature extraction techniques and machine learning algorithm works best for identifying code smell discussions from SO posts?

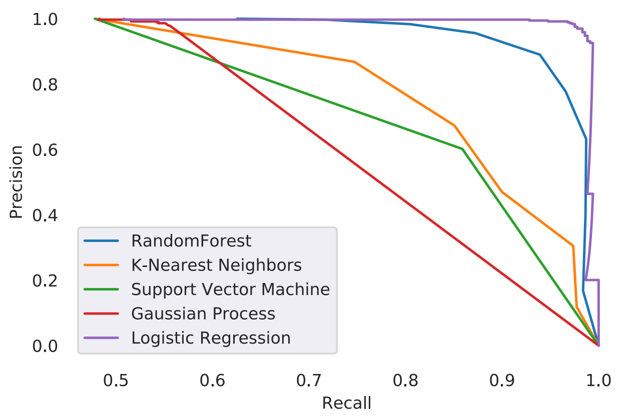

Figure 2: Precision-Recall curve of various algorithms with BoW

Table 3 shows the initial results of evaluating different feature extraction techniques, in which BoW and TF-IDF perform better than Word2Vec, Doc2Vec, FastText, and GloVe. We thus used those two techniques (BoW and TF-IDF) **to answer RQ2**. In particular, we calculated the performance obtained using all configurations, i.e., 28 configurations for BoW and TF-IDF (with *n-gram* equals 1, 2, 3) combined with different values of parameter *C* (other hyper-parameters, such as penalty, solver, work better by default values). As shown in Table 4, the highest F1-score (0.971) is achieved by BoW and LR when *C* equals 20. In fact, we can observe that BoW performs better than TF-IDF in most of the configurations (in terms of F1-score) across different *C* values, with only one exceptional lowest F1-score value being 0.913 when *C* equals 0.001.

Table 3: The initial results of evaluating different feature extraction techniques

	Accuracy	Precision	Recall	F1-score
Word2Vec + LR	0.852	0.883	0.815	0.847
Doc2Vec + LR	0.591	0.637	0.550	0.590
FastText + LR	0.558	0.716	0.402	0.514
GloVe + LR	0.863	0.916	0.824	0.868
BoW + LR	0.970	0.978	0.962	0.970
TF-IDF + LR	0.952	0.963	0.944	0.953

As we can see, for TF-IDF, changing the *n-gram* parameter from two to three does not significantly change the results. At the same time, when using BoW, there is potentially a strong dependency on the regularization coefficient (*C*) at which the result (F1-score) can vary (in our case, from 0.913 to 0.971). An example of a post that was correctly identified by our classifier (TP) is shown below[9]:

"*What are the key anti-patterns to avoid when architecting applications for the enterprise? We are using C# and SQL Server and Silverlight, btw - but I imagine some of the anti-patterns will be language neutral.*"

However, as we can see from the recall and precision levels, there is a small chance that posts will be falsely classified as *code smell* related (FN), such as this example[10].

Summing up the results of both steps, we can say that even when using a relatively small dataset, it is possible to achieve good results when automatically identifying relevant *code-smell* discussions.

We also measured the performance of each algorithm in terms of the execution time required to transform text to vectors and classify the posts. For the testing set (1980 posts), the combination of BoW and LR takes around 8μs to transform posts into vectors and classify them. This also applies to the other algorithms (running being very short). Therefore, there is no noticeable performance overhead when running any of these algorithms on our dataset.

Table 4: F1-score of LR with BoW and TF-IDF

C value		0.001	0.01	0.1	1	10	20	30
BoW		0.913	0.968	0.970	0.970	0.969	0.971	0.970
	n-gram=1	0.946	0.950	0.952	0.949	0.948	0.942	0.943
TF-IDF	n-gram=2	0.957	0.963	0.960	0.960	0.953	0.947	0.947
	n-gram=3	0.957	0.960	0.961	0.959	0.951	0.939	0.942

5 THREATS TO VALIDITY

Internal Validity: A potential threat in this study is the manual labelling of the dataset used in the experiment. The *code-smell* and *non-code-smell* discussions used in the experiment were manually labelled, which may introduce selection bias. It is possible that some posts were incorrectly labelled, leading to incorrect interpretation of the results. The first part of the dataset (*code-smell* discussions) was labelled in [26], which followed a rigorous approach for filtering and manually labelling the data, and involved multiple coders and was performed in multiple stages. The second part of the dataset (*non-code-smell* discussions) was randomly selected from SO and manually labelled by one author with cross-validation by two authors to reduce potential bias.

External Validity reflects the extent to which the study results and findings can be generalized in other cases with similar characteristics. A potential threat is the fact that the data obtained only from SO. Therefore, the generalization of our classification model can be limited to SO. This threat can be partially alleviated by including the data from other forums, QA sites, and textual artefacts (e.g., issues and code reviews).

Construct Validity in this work focuses on whether the evaluation metrics are suitable and measured correctly. A set of metrics (i.e., accuracy, recall, precision, and F1-score) were used to measure the performance of the classification algorithms, which have been widely used in assessing the quality of information retrieval tasks in software engineering.

Reliability refers to whether the experiment yields the same results when it is replicated. In this work, this validity is mainly related to the process of manual labelling of the dataset and the execution of the experiment. We defined the protocol for the labelling

[9]https://stackoverflow.com/questions/2039789

[10]https://stackoverflow.com/questions/52830

process and specified the classification process and evaluation metrics, which were confirmed and followed by all the researchers. We also made our dataset available for replication purposes [23].

6 CONCLUSIONS AND FUTURE WORK

In this work, we used supervised machine learning algorithms and several feature extraction techniques to automatically identify posts that contain *code-smell* related discussions. First, we created a balanced dataset that contains 3000 *code-smell* posts and another manually labelled 3000 *non-code-smell* posts from SO. Then, to find the best algorithm to classify *code-smell* related posts, we chose five algorithms (RF, kNN, SVM, GP, and LR), which we then evaluated their performance using accuracy, precision, recall, and F1-score. Using the default parameters for all the algorithms with BoW as a feature extraction technique, we found that LR performs significantly better than other algorithms (with accuracy = 0.970, precision = 0.978, recall = 0.962, and F1-score = 0.970).

To find the best parameter and the best feature extraction technique that works with LR, we investigated the performance of LR with different C parameters and six feature extraction techniques (BoW, TF-IDF, Word2Vec, Doc2Vec, FastText, and GloVe). We found that the best algorithm is LR with C equals 20 and BoW as the feature extraction technique. The F1-score of this configuration is 0.971 with a precision of 0.978 and a recall of 0.965.

In the next step of this ongoing work, we plan to (1) evaluate the best configuration of each algorithm on a larger dataset of discussions from SO and other textual artefacts (e.g., issues and code reviews), and also consider other topics that are discussed on these forums; (2) use automatic techniques to identify sub-topics from the *code-smell* discussions, and capture the context of these discussions; (3) evaluate the utility of the classifier by classifying unseen SO posts into *code-smell* discussions, and manually checking the output.

ACKNOWLEDGMENTS

This work has been partially supported by the National Key R&D Program of China with Grant No. 2018YFB1402800 and IBO Technology (Shenzhen) Co., Ltd., China.

REFERENCES

[1] M. Abbes, F. Khomh, Y. G. Gueheneuc, and G. Antoniol. 2011. An empirical study of the impact of two antipatterns blob and spaghetti code on program comprehension. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 181–190.

[2] M. Allamanis and C. Sutton. 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 53–56.

[3] M. Asaduzzaman, C. K. Roy, and K. A. Schneider. 2020. CAPS: a supervised technique for classifying stack overflow posts concerning API issues. *Empirical Software Engineering* 25(2) (2020), 1493–1532.

[4] K. Bajaj, K. Pattabiraman, and A. Mesbah. 2014. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, 112–121.

[5] A. Barua, S. W. Thomas, and A. E. Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19(3) (2014), 619–654.

[6] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. 2014. Are test smells really harmful? an empirical study. *Empirical Software Engineering* 20(4) (2014), 1052–1094.

[7] S. Beyer, C. Macho, and M. Di Penta. 2018. Automatically classifying posts into question categories on stack overflow. In *Proceedings of the 26th International Conference on Program Comprehension (ICPC)*. ACM, 211–221.

[8] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger. 2020. What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering* 25(3) (2020), 2258–2301.

[9] S. Beyer and M. Pinzger. 2014. A manual categorization of android app development issues on Stack Overflow. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 531–535.

[10] S. Bird, E. Klein, and E. Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit.* O'Reilly Media.

[11] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[12] M. Fowler. 2018. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional.

[13] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovi. 2009. Identifying architectural bad smells. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 255–258.

[14] F. Khomh, M. Di Penta, Y. G. Guéhéneuc, and G. Antoniol. 2012. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empirical Software Engineering* 17(3) (2012), 243–275.

[15] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, et al. 2006. Handling imbalanced datasets: a review. *GESTS International Transactions on Computer Science and Engineering* 30(1) (2006), 25–36.

[16] J. H. Lau and T. Baldwin. 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. *preprint arXiv:1607.05368* (2016).

[17] S. M. Olbrich, D S Cruzes, and D. I. Sjøberg. 2010. Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems. In *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1–10.

[18] D. N. Palacio, D. McCrystal, K. Moran, C. B. Cardenas, D. Poshavank, and C. Shenefiel. 2019. Learning to identify security-related issues using convolutional neural networks. In *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 140–144.

[19] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, and A. De Lucia. 2014. Do they really smell bad? A study on developers' perception of bad code smells. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 101–110.

[20] J. Pennington, R. Socher, and C. D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 1532–1543.

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

[22] F. A. Shah, K. Sirts, and D. Pfahl. 2019. Simple app review classification with only lexical features. In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT)*. SciTePress, 146–153.

[23] S. Shcherban, P. Liang, A. Tahir, and X. Li. 2020. Dataset of the Paper "Automatic Identification of Code Smell Discussions on Stack Overflow: A Preliminary Investigation". https://doi.org/10.5281/zenodo.3933982

[24] C. Stanik, L. Montgomery, D. Martens, D. Fucci, and W. Maalej. 2018. A simple NLP-based approach to support onboarding and retention in open source communities. In *Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 68–78.

[25] A. Tahir, J. Dietrich, S. Counsell, S. Licorish, and A. Yamashita. 2020. A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites. *Information and Software Technology* 125 (2020), 106333.

[26] A. Tahir, A. Yamashita, S. Licorish, and S. Counsell J. Dietrich. 2018. Can you tell me if it smells?: a study on how developers discuss code smells and anti-pattern in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 68–78.

[27] F. Tian, F. Lu, P. Liang, and M. A. Babar. 2020. Automatic identification of architecture smell discussions from stack overflow. In *Proceedings of the 32nd International Conference on Software Engineering and Knowledge Engineering (SEKE)*. KSI, 451–456.

[28] C. Treude, O. Barzilay, and M. A. Storey. 2011. How do programmers ask and answer questions on the web? In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. ACM, 804–807.

[29] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk. 2015. When and why your code starts to smell bad. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. IEEE, 403–414.

[30] A. Yamashita and L. Moonen. 2013. Do developers care about code smells? an exploratory survey. In *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 242–251.

[31] Y. Zhang, R. Jin, and Z. H. Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1(1-4) (2010), 43–52.