# Automatic Identification of Architecture Smell Discussions from Stack Overflow

Fangchao Tian [1,2], Fan Lu [1], Peng Liang [1,*], Muhammad Ali Babar [2]

[1] School of Computer Science, Wuhan University, Wuhan, China

[2] School of Computer Science, The University of Adelaide, Adelaide, Australia

***Abstract*—Architecture Smells (ASs), as one source of technical debt, indicate underlying problems at a high level of systems and negatively impact various system qualities, such as maintainability and evolvability. Detecting and refactoring ASs requires the relevant architectural knowledge and experience. Therefore, gathering the knowledge of ASs from various sources can facilitate ASs detecting and refactoring. However, manually identifying AS knowledge is time-consuming. Automatically and correctly identifying AS-related posts from Stack Overflow is a step toward utilizing the AS knowledge to help developers better maintain their systems. In this work, we propose an approach to automatically identify AS-related posts from Stack Overflow (SoF) by using machine learning algorithms. We evaluate the performance of 12 classifiers based on 3 feature extraction techniques and 4 classification algorithms with a created dataset of SoF posts (including 208 AS-related posts and 187 AS-unrelated posts). The results demonstrate that the SVM algorithm with Word2Vec achieved the best overall performance with an accuracy of 0.650, a precision of 0.613, a recall of 0.905, and an F1-score of 0.731. These results imply that the obtained model of the AS-related posts identification can be used to aid developers and researchers in collecting AS discussions from SoF.**

***Keywords—Architecture Smell, Architecture Smell Discussion, Stack Overflow, Text Classification***

## I. Introduction

Architecture Smells (ASs) are proposed as frequently recurring architectural decisions that negatively impact system quality [1]. ASs, as the counterpart of code smells, occur at a higher granularity level of a system and can have system-wide impact on maintainability issues. Therefore, detecting and refactoring ASs require more effort compared to code smells [2]. Different researchers defined different categories of ASs with supported detection tools [3]. Fontana *et al.* defined three dependency-related ASs and proposed a tool, called ARCAN, to detect them by analyzing dependency graphs extracted from the packages of compiled Java projects [2]. Mo *et al.* proposed Hotspot Detector to detect five types of ASs, called Hotspot Patterns, defined at the package and file levels [4]. Le *et al.* presented ARCADE which can detect 11 types of ASs across 4 categories [5]. These ASs detection tools are metrics-based and apply some fixed threshold to judge whether a package or component is smelly or not. But it is challenging to manually choose the metrics and thresholds, which can induce false-positive instances of ASs. Correctly detecting and refactoring ASs requires knowledge and experience of developers and researchers to remove false positive instances [2]. Therefore, smell detection and refactoring rely on the knowledge and experience of developers and researchers, and need to consider different aspects such as system domain, software context, and software engineering experience. Moreover, unlike ASs detection, ASs refactoring is less researched and reported in the literature. Empirical studies on investing the knowledge and experience of detecting and refactoring ASs are needed [6].

Stack Overflow (SoF), as a crowdsourced knowledge sharing platform, has been a popularly and widely used software and development Questions and Answers (Q&A) sites that contain more than 18 million questions across a wide variety of topics since 2008 [7]. The knowledge and experience of developers in SoF has been adopted by researchers to study various topics. Tahir *et al.* investigated the developers' perception of code smells and anti-patterns by mining and analyzing the discussions about these two concepts in SoF [8]. In our previous work, we manually collected and analyzed 207 AS-related posts to investigate the understanding of developers about ASs [3], such as the approaches and tools used to detect and refactor ASs. Therefore, the discussions in SoF posts can provide knowledge of ASs and refactoring suggestions that can be used to guide a developer or architect in understanding and addressing potential issues in the architecture of a software system. However, from these studies [3][8], we can find that searching smell-related posts via tags or search terms is ineffective and induces false-positive posts. Furthermore, manually identifying AS-related posts is a time-consuming and subjective process which requires the expertise and experience of ASs and can also lead to inaccurate or incomplete posts. To address this challenge, automatically mining and identifying AS-related posts is needed.

Machine Learning (ML) and Nature Language Process (NLP) techniques have been extensively used to automatically identify or mine meaningful information from SoF posts. For example, Ahasanuzzaman *et al.* built a technique, called CAPS, that can automatically classify SoF posts concerning API issues [9]. Borg *et al.* used active learning to train an SVM classifier for identifying SoF posts concerning the performance of software components [10]. To the best of our knowledge, there is currently no study that automatically mines SoF post discussing ASs. Our research aims at closing this gap by automatically identifying AS-related posts from SoF.

We developed an approach to automate the classification of AS-related posts. We created a dataset, consisting of labelled

* Corresponding author

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.

DOI reference number: 10.18293/SEKE2020-084

208 AS-related posts and 187 AS-unrelated posts, for training classification models. Furthermore, we ran an experiment with 12 configurations by using three feature extraction techniques (i.e., BoW, TF-IDF, and Word2Vec) and four classification algorithms (i.e., LR, SVM, KNN, and RF). We then compared the performance of the models measured in terms of accuracy, precision, recall, and F1-score to determine the best configuration. In our experiment, the use of the SVM algorithm with Word2Vec performed best for automating the classification of AS-related posts.

Thus, our paper makes these contributions: (1) a manually labelled dataset consisting of 208 AS-related posts and 187 AS-unrelated posts; (2) an approach to automatically identify AS-related posts using 12 different configurations regarding 3 feature extraction techniques and 4 classification algorithms; (3) an evaluation of the performance of 12 different classifiers on the dataset.

The rest of this paper is organized as follows. Section II presents the related work. The experiment methodology is explained in Section III. The results of our experiment are reported and discussed in Section IV. Section V concludes this work with future directions.

## II. Related Work

In the last years, research and practice on ASs has gained significant attention [2]. In this section, we provide an overview of ASs and automatic techniques for mining textual information from Stack Overflow.

### A. *Architecture Smells*

Several studies proposed the definitions of ASs with different subtypes. ASs was originally proposed by Lipper [11] to indicate the underlying problems that occur at the architecture level of a system. They also provided a catalogue of ASs at different levels: dependency graphs, inheritance hierarchies, packages, subsystems and layer. Some of these ASs were provided with refactoring measures. Garcia et al. considered ASs as instances of poor architecture decisions that can affect a system life cycle properties, such as understandability and testability [1]. Moreover, they described four types of ASs and each smell's impact on a system lifecycle properties. Fontana et al. presented an ASs Detector, called ARCAN, which can identify three different dependency based ASs: Unstable Dependency, HubLike Dependency and Cyclic Dependency [2]. They later developed a prototype tool, as an extension of the Arcan tool, which can provide refactoring suggestions to remove Cyclic Dependency smell [12]. In another study, Mo et al. formally defined five architecture hotspot patterns and presented a tool, called hotspot detector, to automatically detect and identify these smells at packages or files level [4]. Based on these works [1][4][6], Le *et al.* reviewed and integrated previously reported ASs. Finally, they described 11 ASs and classified them into four categories. More importantly, all these 11 ASs can be automatically detected by the proposed ARCAN and the corresponding detection algorithm [5].

As mentioned by Fontana *et al.* [2], even the detection tools can induce false-negative AS instances, which require additional effort and experience as well as a better understanding of the smells to avoid false-positives instances. As reported in our previous study [6], SoF, as an online community for sharing knowledge, can provide a rich knowledge and experience about AS understanding, detection, and refactoring.

### B. *Mining Information from Stack Overflow*

Many studies have been performed to automatically mine SoF data from different perspectives using ML or NLP. Karthik *et al.* developed an automated mechanism using an unsupervised deep learning based method to identify three different types of compatibility relations between components from the unstructured text on Q&A site postings [13]. Beyer *et al.* built a classification model using ML algorithms (Random Forest and Support Vector Machines) to automatically classify SO posts into seven question categories [14]. In another study, Borg *et al.* made an attempt to use Active Learning and an SVM classifier for mining performance discussions on SoF posts with two alternating annotators [10]. Zhang *et al.* investigated an approach using NLP and sentiment analysis techniques to automatically extract problematic API features from SoF posts [15]. Furthermore, Ahasanuzzaman *et al.* presented a supervised learning approach using Conditional Random Field (CRF) to identify API issue-related sentences in an SoF post [9].

However, none of the works above focuses on the classification of AS posts in SoF. Inspired by the existing works, we plan to use ML and NLP techniques to automatically identify AS discussions on SoF posts.

## III. Research Design

In this section, we describe the goal and Research Questions (RQs), and the method used in the study design.

### A. *Research Questions*

The objective of our work is to provide an approach to automatically mine and identify AS discussions from textual artefacts. To achieve this objective, we define the following three RQs and explain their rationale.

**RQ1: Which technique (BoW vs. TF-IDF vs. Word2Vec) performs best in the feature extraction step when identifying AS-related posts from SoF?**

**Rationale**: In text identification tasks, Text Data Vectorization is an essential process that converts text data into a set of real numbers (a vector). We use four well-performed vectorization methods for extracting textual features: BoW, TF-IDF, and Word2Vec. BoW (Bag of Words), as one of the most commonly used traditional vector representations, links each word or n-gram to a vector index that represents weather word occurs in a document or not. TF- IDF is a statistical measure used to evaluate the importance of a word to a document in a collection of documents or corpus. Word2Vec, introduced by Google, is a predictive embedding model to produce a distributed representation of words with word semantics [16]. There are two main models of Word2Vec - Continuous Bag of Words (CBOW) and Skip-Gram. Employing different vectorization techniques may affect the final performance of classifiers. Therefore, the aim of the question is to determine the vector representation which can achieve the best performance when identifying AS discussions from SoF post.

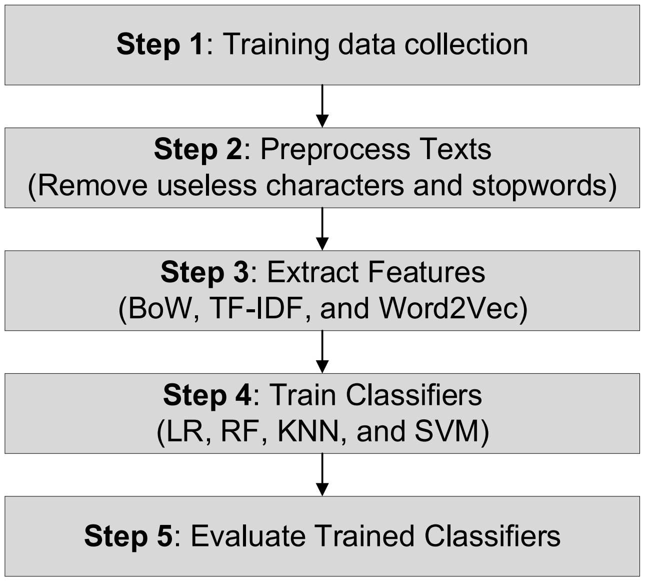
**Step 1**: Training data collection
**Step 2**: Preprocess Texts
(Remove useless characters and stopwords)
**Step 3**: Extract Features
(BoW, TF-IDF, and Word2Vec)
**Step 4**: Train Classifiers
(LR, RF, KNN, and SVM)
**Step 5**: Evaluate Trained Classifiers

Fig. 1.The overall process of classifying AS-related posts

**RQ2: Which classification algorithm (i.e., LR, SVM, KNN, and RF) performs best when identifying AS-related posts from SoF?**

**Rationale**: Various text classifiers have been employed in the literature based on ML techniques, probabilistic models, etc. Different classification methods may lead to differences in classification performance when coping with text identification tasks [18]. We use four commonly used classification algorithms (i.e., Random Forest (RF), KNN, SVM, and Logistic Regression (LR)) for classifying textual artifacts in software development (e.g., [17]) and compare their performance in our AS-related posts identification tasks. By answering this question, we can determine the kind of classification algorithm that can perform best in automatically identifying AS-related posts.

**RQ3: What is the best configuration to automatically identify AS-related posts from SoF by combining different feature extraction techniques and classification algorithms?**

**Rationale:** Different performances can be achieved by using different feature extraction techniques and classification algorithms. We used three feature extraction techniques and four classification algorithms, which results in 12 classifier configurations. The configuration with best overall performance may not be combined by the technique and algorithm which achieve the best performance in each separate step. Therefore, the aim of this RQ is to analyze the performance of classifiers with different configurations and determine the best configuration (that achieves the best performance) for identifying AS-related posts.

## B. *Study Design*

In this section, we introduce how we performed an experiment to identify AS discussions from SoF posts using automatic techniques. As shown in Fig. 1, the study design consists of five steps. In the following, we describe the details of the tasks conducted in each step.

**Step 1: Training data collection**. The input of the classification process of AS-related posts in Fig. 1 is SoF posts. We used a set of inclusion and exclusion criteria (enlisted in TABLE I) [3] for manually selecting and labeling AS posts. If a post contained at least one sentence which met one criterion, we labeled this post as AS-related. Furthermore, we used the criterion C7 to label AS-unrelated posts. Each post was independently analyzed and manually labelled as AS-related or unrelated by two of the authors. To mitigate unconscious bias, any disagreements of the labelling results were discussed and resolved with the help of a third author.

We selected and labelled 395 posts (208 AS-related posts and 187 AS-unrelated posts) according to the above criteria. A few examples of AS-related and AS-unrelated posts are provided in TABLE II. This dataset was then split into two parts: (a) 90% of the posts as the training data set, and (b) 10% of the posts as the testing data set. To support preprocessing in the next step, we used a web crawler to collect and store the content of the labelled SO posts by using their URLs. The crawler works in four steps: (1) extract the URLs of the labelled posts, (2) remove useless URL information in the posts (e.g., *http://www.xxx.org/*), (3) parse and extract post information (i.e., titles, questions, and answers), and (4) save the post information into a CSV document. For the replicability of our experiment, the dataset of our experiment has been made available online[1].

TABLE I. Criteria for Labelling AS-Related Sentences

| | **Criterion ID** | **Description** |
|---|---|---|
| **Criteria for labelling AS-related sentences** | C1: Description of Ass | Descriptions of ASs by practitioners based on their understanding |
| | C2: Cause of ASs | Causes that lead to ASs |
| | C3: Approach for detecting and refactoring ASs | Methods used to detect/refactor specific ASs (e.g., machine learning based approaches) |
| | C4: Tool for detecting and refactoring ASs | Tools for detecting and refactoring specific ASs (e.g., source code analysis tools to identify dependency cycles) |
| | C5: Impact of Ass | Impact of ASs on software development (e.g., understandability, testability, extensibility, and reusability) |
| | C6: Challenge of detecting and refactoring ASs | Challenges identified in detecting and refactoring ASs |
| **Criterion for labelling AS-unrelated sentences** | C7: Not AS-related topic | The sentence does not describe ASs or the sentence topic is not about AS, for example sentences do not describe AS but only other types of smells (e.g., code smells). |

[1] https://tinyurl.com/wdhy46l

TABLE II. Examples of AS-related Sentences and AS-unrelated Sentences

| Type | Example |
|---|---|
| **AS-related Sentence** | "*Is using a root persistent class or base persistable object an architecture smell?*" |
| | "*I think my architecture has kind of a smell to it: The webservice is acting as a proxy, collecting information from different sources.*" |
| | "*Message Bus and Message Based Architecture With Winforms/Desktop Application and Strategies/Policies for View/UI Logic*" |
| | "*What would be a nice architecture so I can pass information of eventual problems to a higher layer?*" |
| | "*Using a command architecture is a good idea, since this moves all business logic out of the controller, and allows you to add cross-cutting concerns without changes to the code.*" |
| **AS-unrelated Sentence** | "*There's a distinct smell of burned out circuits coming from my head, so forgive my ignorance.*" |
| | "*For some derived classes, I want to ensure that one of two overloaded abstract methods get overridden, but not both. Is this possible?*" |
| | "*Judging by the quality of the pixels that have been restored properly, the network architecture seems to be fine for this task.*" |
| | "*I came across the Open Test Architecture API and was wondering if there are any good Python or java examples for the same that I could see.*" |

**Step 2: Data Preprocessing**. Data preprocessing eliminates the terms or characters in the training posts that are unnecessary to train classifiers for identifying AS discussions, which is composed of 2 steps: (1) **Removing useless characters**. Since the posts were crawled from Stack Overflow website which is formed in HTML 5, most posts contain some useless punctuations like "…" and escape characters like "/n" or "/r". Those characters provide invalid information in semantic parsing, so we removed those useless characters. (2) **Processing stop words**. Referring to the original idea of TF-IDF, daily language interaction like Q&A posts from Stack Overflow can be filled with common words such as auxiliary verbs, conjunctions and articles. Removing stop words can reduce the noise in natural language, because these words also lack the distinguishing feature for training classifiers. To remove those meaningless words, we apply the default stop words list in Natural Language Toolkit (NLTK) package.

Note that, in our study, we did not apply stemming and lemmatizing but stop words removing to preprocess the training data, because stemming and lemmatizing may change the meaning of the text. Moreover, prior research shows that the text preprocessing method with No Stemming and Lemmatization performs best when preprocessing posts to identify decisions from textual artifacts in software development [17].

**Step 3: Feature extraction**. The aim of feature extraction is to transform preprocessed documents into numerical vector representations for classifiers regardless of the vectorization method. In our work, we applied three feature extraction techniques, i.e., BoW, TF-IDF, and Word2Vec to calculate the feature value of each post.

BoW is a commonly used traditional feature extraction technique. A corpus is created consisting of every unique word across the documents. Then each word is converted into the corresponding vector by counting the occurrence of a word in a document. While BoW is simple to understand and implement, it lacks the ordering of words, which leads to loss of contextual information and word meaning in the document (semantics).

TF-IDF is a basic vectorization method which considers frequencies of words in one document and words relationships among documents. However, TF-IDF is also unable to capture the word meaning. TF-IDF produces vectors based on frequencies of words in one document (TF) and the weight of rare words across all documents (IDF). TF-IDF used in this study is defined as in Formula (1):

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i}) \tag{1}$$

$tf_{i,j}$ represent the count of a term "I" in a document "j", N represents the number of total documents in the corpus, and $df_i$ represents the number of total documents containing the term i.

Word2Vec is a word embedding method which produces dimensional numerical representations of words and more syntactic information than BoW and TF-IDF. These two models of Word2Vec: Continuous Bag of Words (CBOW) and Skip-Gram. The CBOW model obtains word representations by predicting the current word based on its context (surrounding words). Contrary to the CBOW model, The Skip-Gram learns the embedding by predicting the surrounding words (context) given a current word. CBOW is several times faster than Skip-Gram, while Skip-Gram performs better for even rare words or phrases than CBOW.

**Step 4: Classifier Training**. After transforming the collected SoF into numerical vectors, we used the extracted features to train four algorithms, i.e., LR, SVM, RF, and KNN, to automatically identify AS-related posts. RF, KNN, and SVM are three non-parametric classifiers. In contrast, LR is a parametric classifier and faster and simpler classification method than the other three. We used the implementation of these four classifiers in the scikit-learn Python package.

**Step 5: Performance Evaluation**. To evaluate the performance of three feature extracting methods and four classifiers, we used four common measures: accuracy, precision, recall, and F1-score. We used Formula (2), (3), (4), and (5) to calculate accuracy, precision, recall, and F1-score, respectively.

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

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

$$F1-score = \frac{2*Precision*Recall}{Precision+Recall} \tag{5}$$

Among these formulas, True Positive (TP) denotes the number of correctly identified AS-related posts by classifiers. False Positive (FP) represents the number of AS-unrelated posts that are incorrectly identified as AS-related posts by classifiers. False Negative (FN) indicates the number of AS-related posts that are incorrectly labelled as AS-unrelated posts by classifiers. True Negative (FN) shows the number of correctly identified AS-unrelated posts by classifiers.

Therefore, in our context, precision is used to measure the exactness of prediction set and represented as the ratio of posts correctly identified as AS-related posts to all posts identified as AS-related posts. Recall is the fraction of all AS-related posts correctly demarcated. F1-score is a combination of precision and recall. We consider precision and recall equally important. F1-score is calculated by the harmonic mean of precision and recall. Accuracy is the ratio of correctly identified posts, including AS-related posts (TP) and AS-unrelated posts (TN) to all identified posts.

## IV. Results and Analysis

As described in Section III, we conducted an experiment with 3 (three feature extraction techniques) × 4 (four classification algorithms) = 12 configurations. We calculated these metrics (i.e., precision, recall, F1-score, and accuracy) to evaluate the performance of each configuration for identifying AS-related posts in SoF. In this section, we present and analyze the results to answer the research questions.

### A. *Results and Analysis of RQ1*

To answer RQ1, we calculated the average of the results obtained by three feature extraction techniques to observe the impact of different techniques on the performance of classifiers. TABLE III shows the performance of ASs classifiers by employing three feature extraction techniques (i.e., TF-IDF, BoW, Word2vec). The highest accuracy, precision, recall, and F1-score values of AS discussion classification across all techniques are highlighted in boldface. we found that TF-IDF technique can achieve the highest average precision (0.620), accuracy (0.619), and F1-score (0.714), and the BoW technique can achieve the best Recall (0.798). One important observation is that the recall value for the TF-IDF is very poor. It can also be found that the performance of the Word2Vec model is poor, which is consistent with the result observed by Li *et al.* [17]. The results indicate that the TF-IDF technique can perform better than the other techniques (i.e., Bow, Word2Vec) and extract meaningful AS discussions from SoF posts.

TABLE III. Average Results of Different Feature Extraction Techniques for Identifying Architecture Smell Discussions

| Technique | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| TF-IDF | **0.619** | **0.620** | 0.698 | **0.714** |
| BoW | 0.516 | 0.528 | **0.798** | 0.629 |
| Word2vec | 0.556 | 0.563 | 0.631 | 0.589 |

### B. *Results and Analysis of RQ2*

To answer RQ2, we calculated the average of the results obtained by the four classification algorithms to observe the impact of different algorithms on the performance of classifiers (i.e., the results of AS discussions classification). TABLE IV presents the performance of ASs classifiers by employing the four classification algorithms (i.e., LR, SVM, RF, KNN). The highest precision, recall and F1-score of AS identification across all classification algorithms are highlighted in boldface. We observed that the highest average accuracy (0.600), recall (0.921) and F1-score (0.709) are achieved by the SVM-based classifier, and the highest average precision (0.589) are achieved by KNN-based algorithm. Moreover, SVM can achieve the second-highest average precision (0.579). The performance of KNN-based classifier is slightly worse than SVM, with an accuracy of (0.592), a recall of (0.762), and an F1-score of (0.620). DT-based classifier got the lowest average precision (0.583), recall (0.577), and F1-score (0.577). These results indicate that the SVM-based classifier performs the best in term of the overall performance and can be used as the most suitable classification algorithm in the classifier training step when automatically identifying ASs from SoF compared to other three classification algorithms. This conclusion is consistent with the findings reported in [17].

TABLE IV. Average Results of Different Classification Algorithms for Identifying Architecture Smell Discussions

| Algorithm | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| LR | 0.542 | 0.560 | 0.571 | 0.566 |
| SVM | **0.600** | 0.579 | **0.921** | **0.709** |
| RF | 0.525 | 0.553 | 0.603 | 0.570 |
| KNN | 0.592 | **0.589** | 0.762 | 0.620 |

### C. *Results and Analysis of RQ3*

To answer RQ3, we calculated and compared the precision, recall, and F1-score of each of 12 configurations to analyze how accurately we can automatically identify AS-related posts from SoF. TABLE V presents the performance of 12 ASs classifiers by employing four classification algorithms (i.e., LR, SVM, RF, KNN) across three feature extraction techniques. The highest precision, recall, and F1-score values for ASs classifiers across all configurations are highlighted in boldface. It can be found that the highest accuracy (0.650), precision (0.636), recall (1.000), and F1-score (0.731) can be achieved with the combination of three feature extraction techniques and four classification algorithms. We can also find that there are two configurations with an F1-score more than 0.7 and accuracy greater than 0.625. It is also worth noting that these two configurations employ SVM. The highest precision (0.650) and F1-score (0.731) are achieved by the combination of Word2Vec with SVM, which confirms the result of RQ2 that SVM-based classifier performs the best for automatic classification of AS-related posts. One important observation is that although the TF-IDF technique can achieve the best performance in the step of text feature extraction when identifying AS-related posts, the combination of Word2vec and SVM can achieve the best overall performance. These results indicate that each configuration presents a diverse performance. Moreover, the configuration with the best performance is not simply combined by the technique and algorithm which achieves the best performance in each separate step, respectively. Overall, with an accuracy of 0.650, a precision of 0.613, a recall of 0.905, and an F1-score of 0.731, the configuration with a combination of Word2Vec and SVM can achieve the best performance for identifying AS-related posts, and can be used by developers and researchers to collect AS discussions from SoF.

TABLE V. Evaluation Results of Combining Different Feature Extraction Techniques and Classification Algorithms for Identifying Architecture Smell Discussions

| Technique | Algorithm | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| TF-IDF | LR | 0.625 | **0.636** | 0.667 | 0.651 |
| | SVM | **0.625** | 0.600 | 0.857 | **0.706** |
| | RF | 0.600 | 0.619 | 0.619 | 0.619 |
| | KNN | 0.625 | 0.625 | 0.714 | 0.619 |
| BoW | LR | 0.525 | 0.545 | 0.571 | 0.558 |
| | SVM | 0.525 | 0.525 | **1.000** | 0.689 |
| | RF | 0.450 | 0.484 | 0.714 | 0.577 |
| | KNN | 0.575 | 0.559 | 0.905 | 0.691 |
| Word2Vec | LR | 0.475 | 0.500 | 0.476 | 0.488 |
| | SVM | **0.650** | 0.613 | 0.905 | **0.731** |
| | RF | 0.525 | 0.556 | 0.476 | 0.513 |
| | KNN | 0.575 | 0.583 | 0.667 | 0.622 |

## V. Conclusions and Future Work

Architecture Smells (ASs), as a type of technical debt, have been attaining importance in recent years since they may significantly depreciate software quality. Detecting and refactoring ASs correctly require appropriate architectural knowledge. The online communities, such as Stack Overflow (SoF), contain a wealth of latent information expressed in natural language and become a valuable source of information for sharing knowledge of ASs. But manually gathering AS discussions from the online communities is a time-consuming and labor-intensive task. To address this problem, we have proposed a solution to automatically identify AS related posts from SoF using 12 different configurations of feature extraction techniques and classification algorithms.

We first created a dataset from SoF consisting of 208 AS-related posts and 187 AS-unrelated posts and manually labelled AS-related sentences for training classification models across the combinations of the three feature extraction techniques and four classification algorithms. We evaluated and discussed the performance of the 12 combinations by calculating four metrics: accuracy, precision, recall, and F1-score. The results from our experiment show that: (1) the TF-IDF technique performs best when extracting text features to identify ASs; (2) the SVM based classifier achieves the best overall performance regarding accuracy and F1-score of automatic AS discussions classification; and (3) the SVM algorithm with Word2Vec outperforms the other combinations when automatically identifying AS-related posts.

These results provide several implications for our future research including: (1) validating the best configurations in other online sources of textual information such as developer mailing lists and issue tracking systems, (2) to identify AS-related discussions at the sentence level, which provides more focused content about AS discussions, (3) employing multi-classification algorithms (e.g., Softmax) to automatically classify different types of AS-related discussions from textual artefacts, and (4) automatically extracting AS information to enrich practioners' knowledge and experience of detecting and refactoring ASs, and evaluating whether and how the AS information can improve detecting and refactoring ASs.

## References

[1] Garcia, J., Popescu, D., Edwards, G. and Medvidovic, N., 2009. Identifying architectural bad smells. In: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR). Kaiserslautern, Germany. pp. 255-258.

[2] Fontana, F.A., Pigazzini, I., Roveda, R. and Zanoni, M., 2016. Automatic detection of instability architectural smells. In: Proceedings of the 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME). Raleigh, NC, USA, pp. 433-437.

[3] Tian, F., Liang, P. and Babar, M. A., 2019. How developers discuss architecture smells? an exploratory study on Stack Overflow. In: Proceedings of the 16th International Conference on Software Architecture (ICSA). Hamburg, Germany, pp. 91-100.

[4] Mo, R., Cai, Y., Kazman, R. and Xiao, L., 2015. Hotspot patterns: the formal definition and automatic detection of architecture smells. In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), Montreal, QC, Canada, pp. 51-60.

[5] Le, D. M., Link, D., Shahbazian, A., & Medvidovic, N. 2018. An empirical study of architectural decay in open-source software. In: Proceedings of the 15th IEEE International Conference on Software Architecture (ICSA), Seattle, WA, USA, pp. 176-185.

[6] Samarthyam, G., Suryanarayana, G. and Sharma, T., 2016. Refactoring for software architecture smells. In: Proceedings of the 1st International Workshop on Software Refactoring (IWoR). Singapore, Singapore, pp. 1-4.

[7] Grant, S. and Betts, B., 2013. Encouraging user behaviour with achievements: an empirical study. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR). San Francisco, CA, USA, pp. 65-68.

[8] Tahir, A., Yamashita, A., Licorish, S., Dietrich, J. and Counsell, S., 2018. Can you tell me if it smells?: a study on how developers discuss code smells and anti-patterns in Stack Overflow. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE), Christchurch, New Zealand, pp. 68-78.

[9] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K. and Schneider, K.A., 2020. CAPS: a supervised technique for classifying Stack Overflow posts concerning API issues. Empirical Software Engineering, 25: 1493-1532.

[10] Borg, M., Lennerstad, I., Ros, R. and Bjarnason, E., 2017. On using active learning and self-training when mining performance discussions on Stack Overflow. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE), Karlskrona Sweden, pp. 308-313.

[11] Lippert, M. and Roock, S., 2006. Refactoring in large software projects: performing complex restructurings successfully. John Wiley & Sons.

[12] Rizzi, L., Fontana, F.A. and Roveda, R., 2018. Support for architectural smell refactoring. In: Proceedings of the 2nd International Workshop on Refactoring (IwoR), Montpellier, France, pp. 7-10.

[13] Karthik, S. and Medvidovic, N., 2019. Automatic detection of latent software component relationships from online Q&A sites. In: Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), Montreal, QC, Canada, pp. 15-21.

[14] Beyer, S., Macho, C., Di Penta, M. and Pinzger, M., 2018. Automatically classifying posts into question categories on Stack Overflow. In: Proceedings of the 26th International Conference on Program Comprehension (ICPC), Gothenburg, Sweden, pp. 211-221.

[15] Zhang, Y. and Hou, D., 2013. Extracting problematic API features from forum discussions. In: Proceedings of the 21st International Conference on Program Comprehension (ICPC), San Francisco, CA, USA, pp. 142-151.

[16] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[17] Li, X., Liang, P. and Li, Z., 2020. Automatic identification of decisions from the Hibernate developer mailing list. In: Proceedings of the 24st International Conference on Evaluation and Assessment in Software Engineering (EASE), Trondheim, Norway, pp. 51-60.

[18] Ikonomakis, M., Kotsiantis, S. and Tampakas, V., 2005. Text classification using machine learning techniques. WSEAS Transactions on Computers, 4(8): 966-974.