# Automatic Identification of Assumptions from the Hibernate Developer Mailing List

Li, Ruiyin; Liang, Peng; Yang, Chen; Digkas, Georgios; Chatzigeorgiou, Alexander; Xiong, Zhuang

*Published in:*
Proceedings - 2019 26th Asia-Pacific Software Engineering Conference, APSEC 2019

*DOI:*
10.1109/APSEC48747.2019.00060

# Automatic Identification of Assumptions from the Hibernate Developer Mailing List

Ruiyin Li [a,c], Peng Liang [a*], Chen Yang [b], Georgios Digkas [c], Alexander Chatzigeorgiou [d], Zhuang Xiong [a]
[a] School of Computer Science, Wuhan University, 430072 Wuhan, China
[b] IBO Technology (Shenzhen) Co., Ltd., 518057 Shenzhen, China
[c] Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, 9747 AG Groningen, The Netherlands
[d] Department of Applied Informatics, University of Macedonia, Egnatia 156, 546 36, Thessaloniki, Greece
{ryli_cs, liangp, zxiong}@whu.edu.cn, c.yang@ibotech.com.cn, g.digkas@rug.nl, achat@uom.gr

***Abstract*—During the software development life cycle, assumptions are an important type of software development knowledge that can be extracted from textual artifacts. Analyzing assumptions can help to, for example, comprehend software design and further facilitate software maintenance. Manual identification of assumptions by stakeholders is rather time-consuming, especially when analyzing a large dataset of textual artifacts. To address this problem, one promising way is to use automatic techniques for assumption identification. In this study, we conducted an experiment to evaluate the performance of existing machine learning classification algorithms for automatic assumption identification, through a dataset extracted from the Hibernate developer mailing list. The dataset is composed of 400 "*Assumption*" sentences and 400 "*Non-Assumption*" sentences. Seven classifiers using different machine learning algorithms were selected and evaluated. The experiment results show that the SVM algorithm achieved the best performance (with a precision of 0.829, a recall of 0.812, and an $F_1$-score of 0.819). Additionally, according to the ROC curves and related AUC values, the SVM-based classifier comparatively performed better than other classifiers for the binary classification of assumptions.**

***Keywords—Assumption; Automatic Identification; Open Source Software; Hibernate; Mailing List***

## I. INTRODUCTION

As defined in dictionaries[1], an assumption is "*a thing that is accepted as true or as certain to happen, without proof*" or "*a fact or statement taken for granted*". In this work, we adopted the definition of software assumption from a recent industrial case study and a systematic mapping study [1, 2] that "*assumptions are software development knowledge taken for granted or accepted as true without evidence*". The essence of assumptions is uncertainty, i.e., stakeholders believe but they are not sure regarding, for example, the importance, impact, or correctness of a piece of software knowledge (e.g., requirements, design decisions) [1]. For example, "*I'm guessing that what users really need is a command line or GUI tool*" is an assumption about a requirement discussed in the developer mailing list of an open source project. More details about assumptions in software development can be found in Section II.

Assumption is not a new concept in software engineering. Different types of studies on assumptions and their management have been conducted in various fields, including requirements engineering [3-5], software design [6, 7], software construction [8], software maintenance [9], and software evolution [10].

Assumptions and their management are recognized as important in software development by both researchers and practitioners [2], while not well-managed assumptions (e.g., implicit or invalid assumptions) can lead to critical issues in projects (e.g., integration defects, architectural mismatches, and vulnerable systems) [2]. As an example, Lehman *et al.* [9] stated the impact of assumptions on software development: "*even an improvement of 10% in the identification, analysis and correction of assumptions could have saved the economy some £1.7bn, a truly staggering figure that does not take into account loss of life and limb and the economic cost of software failure*". As mentioned by Garlan *et al.* [11], incompatible assumptions can cause architectural mismatches, which is an important concern in software engineering community. Moreover, Steingruebl *et al.* [12] claimed that undocumented assumptions can raise serious problems such as software failures. Therefore, it is beneficial to explicitly identify and record assumptions in software development to improve software maintenance [13], facilitate communication between stakeholders, capture early information regarding design decisions [13], and allow reuse of components [11].

However, during software development, many uncertain descriptions and assumptions are recorded in natural language, such as discussions in developer mailing lists. Manually capturing and identifying assumptions is rather time-consuming and labor intensive, especially when analyzing a large dataset of textual artifacts. Due to the time and cost constraints, using automatic approaches in identifying assumptions is important. In Open Source Software (OSS) community, developers are usually in geographically distributed locations and various time zones. As a result, they communicate knowledge electronically via developer mailing lists (e.g., GroupServer[2], Apache Mailing Archives[3]), which provide a major channel for OSS developers to communicate and discuss various types of development information [14]. Automatically and accurately identifying assumptions from, for example, developer mailing lists is significant for OSS developers to understand the assumptions made during the development, and consequently facilitate the maintenance and evolution of OSS projects. To the best of our knowledge, there

* Corresponding author
[1] http://www.oxforddictionaries.com/definition/english/assumption
[2] http://groupserver.org/groupserver/features/details
[3] https://lists.apache.org/

2640-0715/19/$31.00 ©2019 IEEE
DOI 10.1109/APSEC48747.2019.00060

is no such approaches on automatic identification of assumptions from textual artifacts. To this end, we conducted an experiment to evaluate the performance of existing machine learning classification algorithms for automatic assumption identification, using a dataset extracted from the developer mailing list of Hibernate, which is widely used for database-related communication in Java (especially J2EE) applications.

The contribution of this work is the following: (1) This is the first study focusing on automatic identification of assumptions from developer mailing lists in OSS. (2) We employed Natural Language Processing (NLP) techniques to analyze and extract textual information from the Hibernate developer mailing list. (3) We used seven Machine Learning (ML) algorithms to construct ML-based classifiers and evaluated the performance of the seven ML algorithms through experiments about the binary classification of assumptions. The results show that the Support Vector Machines (SVM) classification algorithm performs better (with a precision of 0.829, a recall of 0.812, and an $F_1$-score of 0.819) than other classifiers.

The rest of this paper is structured as follows. Section II introduces the background and related work on assumptions in software development and automatic techniques for mining textual information. Section III describes the research method used in this study, while Section IV presents details of the experiment. In Section V, we present and discuss the results. Section VI discusses the implications. Section VII discusses the threats to validity. Finally, Section VIII concludes this work and discusses the future research directions.

## II. Background and Related Work

Many studies have been conducted on exploring the impact of assumptions in software development. One important conclusion of those studies is that assumptions play a critical role in the software development life cycle [2]. In this section, we discuss related work on assumptions in software development and automatic techniques for mining textual information.

### A. Assumptions in Software Development

Assumptions exist in the entire life cycle of software development, in which many stakeholders are involved in assumption management and various types of software artifacts are related to assumptions as shown in TABLE I according to our recent mapping study on assumptions and their management in software development [2]. In our previous study [1], we conducted a survey about architectural assumptions and found that architects frequently make assumptions in their work. Meanwhile, architectural assumptions may change (i.e., exhibit dynamic characteristic) over time and they can become invalid or be transformed into other types of artifacts. Moreover, in the process of software development, many problems can be traced to not well-managed assumptions. In the report by Lewis *et al*. [15], the authors stated that "*the potential benefit resulting from this practice may be tremendous in the earlier identification and elimination of design- and requirement-level defects, and in improved change analysis for more predictable and cost-effective software evolution*". Specific examples regarding negative influences of invalid assumptions on software development can also be found in [12].

Tang *et al*. [16] mentioned that assumptions (especially design assumptions) and constraints should be explicitly represented and be used as a context for previous and current architecture decisions. Mamun *et al*. [17] argued that automated checking of invalid assumptions is a big advantage for large-scale and complex system development. Shahin *et al*. [18] also found that assumptions are a major element in a core architecture decision model. Shumaiev *et al*. [19] proposed the feasibility of using NLP techniques to automatically identify various types of uncertain information in software architecture documents, thereby it supports communications of software architecture. As discussed in our previous study [2], uncertainty is a fundamental element in assumptions and we need to treat assumptions and uncertainty as two different but relevant concepts.

Overall, assumptions are commonly found in software development and have close relationships to other types of software artifacts. Yang *et al*. [2] listed the challenges of assumption management and one of them is about distinguishing assumptions from other types of artifacts. This work is motivated by such challenge.

TABLE I. Stakeholders Involved in Assumption Management and Artifacts Related to Assumptions in Development Activities

| Software Development Activity | Stakeholder | Software Artifact |
|---|---|---|
| Requirements engineering | Requirements engineer, Client, Customer, User, etc. | Goal, Use case specification, Feature model, etc. |
| Software design | Architect, Designer, Component designer, etc. | Component, Module, Package, Architecture, etc. |
| Software construction | Developer, Programmer, Component developer, etc. | Source code, Program specification, Class, etc. |
| Software testing | Tester, User, etc. | Bug report, Testing plan, etc. |
| Software maintenance and evolution | Maintainer, etc. | Change requests, Version control information, etc. |

### B. Automatic Techniques for Mining Textual Information

Shumaiev *et al*. [19] analyzed the impact of assumptions on architecture decisions and argued the possibility of automatically retrieving assumptions from textual artifacts and they mention the following: "*using the created corpus, the various machine learning training algorithms proposed by NLP community have to be tested to identify the most accurate one*". Velasco-Elizondo *et al*. [20] proposed a method based on knowledge representation and information extraction to automatically identify suitable patterns from a corpus of architectural pattern descriptions. The results show that the method can help inexperienced software architects to determine whether specific quality attributes are promoted or inhibited, which is useful for pattern selection during architectural design. Furthermore, the method can also be potentially used to extract related information of assumptions.

Textual information classification has also been applied in other types of artifacts. For instance, Pascarella *et al*. [21]

focused on the classification of code comments in Java OSS projects. They investigated how to categorize code comments and proposed an approach to automatically classify the comments. The results show that the classifier could be used as an input for tools that analyze code comments of software systems. Maalej *et al*. [22] applied NLP and ML techniques to automatic classification of apps' user reviews into bug reports, feature requests, user experiences, and ratings. Their findings inspire the design of review analytics tools for helping app vendors and developers to address considerable user reviews, filter reviews, and provide useful information to related stakeholders. The above mentioned methods, techniques, and challenge of distinguishing assumptions from other types of artifacts motivate us to conduct an experiment on automatic identification of assumptions.

## III. RESEARCH METHOD

In this section, we present details of the study design. The goal of this study is formulated through the Goal Question Metric approach [23] as the following: *To analyze textual information in developer mailing lists* ***for the purpose of*** *identification* ***with respect to*** *assumptions in software development* ***from the point of view of*** *OSS developers* ***in the context of*** *OSS application (i.e., Hibernate)*. The study follows the guidelines proposed by Shall *et al*. [24].

### A. Research Questions

The study comprises two Research Questions (RQs). For each RQ, we briefly explain their rationale.

**RQ1: How accurately can we automatically identify assumptions from textual information?**

Generally, in OSS development, developers could be from all over the world. Developer mailing lists act as a major mean of communication in order to discuss development issues, and consequently may contain assumptions and other pieces of knowledge about OSS development. Answering this RQ can provide support for further understanding of assumptions and design decisions in OSS development, and alleviate the problems caused by implicit assumptions. However, there is considerable useless textual information and quotes in the developer mailing lists. This RQ guides us to filter out "*noise*" and obtain useful text representing assumption-related information. Even after filtering out the repeated and useless text, we still face a vast amount of textual information, which can be time-consuming and labor-intensive for identifying assumptions manually. An accurate classifier that could automatically identify assumptions would help to avoid this tedious and monotonous step.

**RQ2**: **Which classification algorithm has the best performance with respect to the automatic identification of assumptions?**

Different ML classification algorithms usually have varying performance in automatic classification problems. For example, perceptron is an algorithm for supervised learning of binary classifiers, which is a type of linear classifier. Naive Bayes classifiers are a cluster of probabilistic classifiers, which are also popular in text categorization since 1960. Moreover, Support Vector Machines (SVM) are widely-used supervised ML algorithms that are utilized to analyze data for classification and regression analysis. By answering this question, we seek to understand which kind of classification algorithm is the most appropriate one to facilitate automatic identification of assumptions.

### B. Study Design

We designed an experiment to answer the RQs, which can be divided into two steps: (1) **Data Processing** and (2) **Classifier Training**, which are shown in Figure 1.

**Step 1: Data Processing**

**1.1 Data Preprocessing**. There are quite a number of useless textual characters in the developer mailing list posts that need to be removed, such as "-------", "***", and "&&". In addition, some redundant characters (e.g., redundant space, line break) and extra quotes (e.g., replies) need to be removed to ensure that the experiment dataset does not have any repeated textual information. We consider textual information without any repeated segments and useless characters as valid. After data preprocessing, valid textual information can be extracted from the developer mailing list posts.

**1.2 Manually Labeling Posts**. To avoid personal bias, the first (a PhD student) and sixth (a master student) authors manually identified and labeled assumptions from the preprocessed textual information in the posts independently and the second author (a senior researcher) reviewed the results. To mitigate unconscious bias, the three authors discussed together to address the conflicts. In this way, we can make sure that the labeled samples are reliable. The output of this step is labeled posts that contain *assumptions* (i.e., "*Assumption*" posts).

TABLE II. EXAMPLES OF ASSUMPTIONS AND NON-ASSUMPTIONS

| Type | Example |
|---|---|
| **Assumption** | "*Dynamic mapping could be a requirement for some developer and programmatic change the mapping could be useful for them.*" |
| | "*It might turn out to be a large enough subset for some people's requirements.*" |
| | "*Our goal might be available to use the code generator to generate the javabean classes defined in the mapping file, as well as many of the other classes, webpages, etc.*" |
| | "*it might be wiser to create a Connection object that switches based on the DAO implementation.*" |
| | "*it would be nice to be able to store database indexes in the mapping files so the SchemaExport (and update) tool can generate those as well.*" |
| **Non-Assumption** | "*The original reason for its existence was that it actually declared two methods.*" |
| | "*It was a simple mistake that would have been picked up if that code had been covered by the tests.*" |
| | "*At deployment time have a tool to generate proxy classes that inherit the business classes and override every public, package and protected method with a method that calls the initialization code when needed and then invokes super.*" |
| | "*But it is common practice that all libs that are needed to build are included in cvs, which makes a lot of sense, because sometimes code changes need a new version of a lib too.*" |
| | "*This approach has the single advantage that it permits caching in the persistence layer.*" |

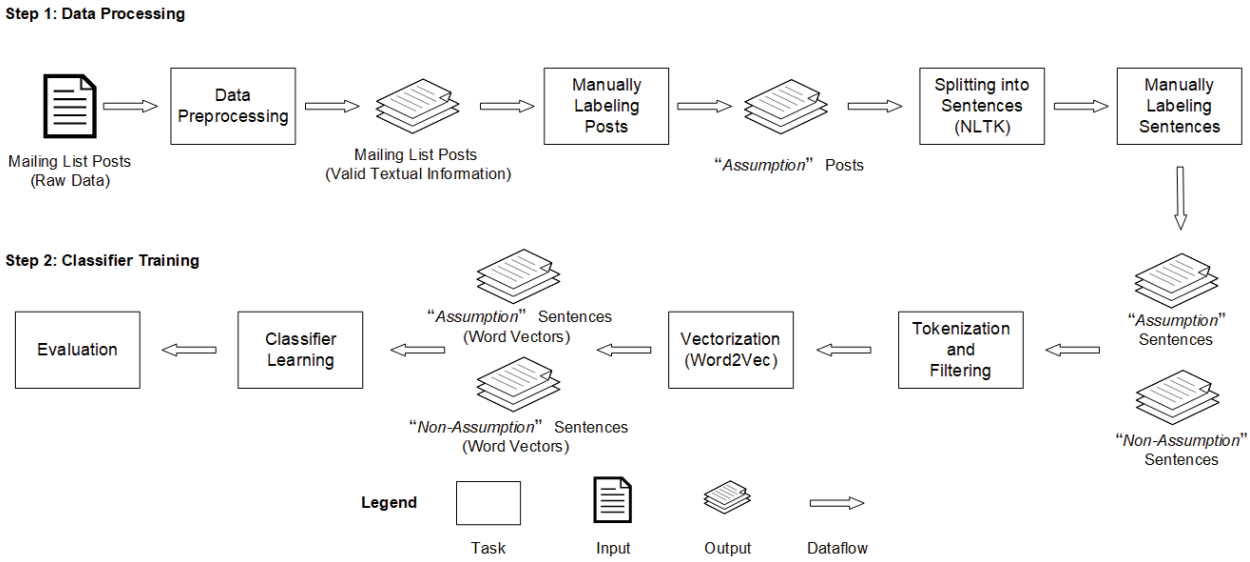


Fig. 1 The overall process of our experiment

**1.3 Splitting into Sentences and Manually Labeling Sentences**. Since the goal of the study is to identify assumptions at the sentence level (see Section IV.A), we employed the *sent_tokenize* tool (from the NLTK toolkit [25] in Python) to split labeled assumption textual segments into individual sentences, and further labeled these sentences as "*Assumption*" or "*Non-Assumption*" respectively as the input of Step 2. We provide a few examples of "*Assumption*" and "*Non-Assumption*" sentences in TABLE II. The criteria for labeling assumptions are detailed in Section IV.

**Step 2: Classifier Training**

Valid textual information is processed to generate structural data for further analysis. For each sentence with the "*Assumption*" label, we extracted features (see Step 2.1) for textual classification. These features are used to train classifiers in order to automatically identify assumptions.

**2.1 Tokenization and Filtering**. All the preprocessed sentences were transformed into bunches of constituent words (i.e., splitting each individual sentence into a string of words). To remove English stop words from each string of words, we used the stop words list provided by the *scikit-learn* tool [26]. To extract stems of words in each sentence, we conducted a stemming process, which reduced inflected forms of words to their basic or root stems that can be extracted or analyzed as a single item. The tool used for stemming is *SnowballStemmer* from the NLTK toolkit, because it could provide better stemming results compared to other tools, such as *Porter* and *Lancater* [27]. From the results of our contrast experiment, *SnowballStemmer* gives considerably more reliable results.

Subsequently, we set a threshold to filter out the word strings that contain less than four words. The reason is that such word strings have limited information and they can be considered as invalid features. We also filtered out those non-ASCII and non-English characters, which were also regarded as meaningless features. The rest of the valid word strings were selected as the features of each sentence.

**2.2 Vectorization**. After tokenization and filtering, we conducted a vectorization process for the selected word strings and transformed them into word vectors through using the Word2Vec tool[4]. Word2Vec is a widely used tool in the NLP community, which can generate word embeddings with rich syntactic information [28]. After the vectorization process, the features of sentences can be used for training classifiers.

**2.3 Classifier Learning**. Based on the processed dataset from the previous steps, we trained seven binary classifiers that can be used to automatically identify assumptions from textual information. The seven binary classifiers treated the "*Assumption*" sentences as positive instances and the "*Non-Assumption*" sentences as negative instances. Since imbalanced problems of training dataset are often present in binary classification, we oversampled the instances and balanced the experiment dataset by selecting the same amount of positive and negative instances. Seven popular supervised ML classification algorithms were employed for the classifiers.

**2.4 Evaluation**. Finally, we evaluated the performance of the seven classifiers. The results and analysis of the evaluation are provided in Section V.

## IV. Experiment

### A. Data Selection

Due to the dynamic nature of assumptions [29] (e.g., a valid assumption can turn out to be invalid, or transform to other types of artifacts), it is hard to verify whether a statement is an assumption only according to the content of each post. Therefore, we decided to first identify assumptions at the sentence level. In our previous work [30], we identified and

[4] https://code.google.com/archive/p/word2vec/

collected 843 posts that contain assumptions from 9006 posts in the Hibernate[5] developer mailing list between Jan 2002 and Jun 2015. The following criteria were used to label assumptions:

(1) An assumption sentence is supposed to be uncertain, which means if there is a strong evidence to support its certainty and validity, it is not an assumption. However, we should note that assumptions are not equal to uncertainties, but rather derived from uncertainties [2].

(2) Consider the context but not just the content. In our previous systematic mapping study [2], we identified many examples of assumptions from the selected studies (e.g., "*If thread i holds the lock in read mode, then x cannot be changed by another thread*" [31]). It is difficult to treat such example as an assumption by only reading such statements [2]. However, if we dig deeper, paying attention to the context of those examples, it becomes clear why the examples should be considered as assumptions, instead of other types of artifacts, such as requirements.

(3) Uncertainty may also be a characteristic of other types of software artifacts, but such characteristic is not the focus. For example, when discussing a design decision, stakeholders usually pay attention to the problems behind, instead of whether the decision is uncertain. On the contrary, assumptions are made to address uncertainties [2].

(4) Note the difference between the content of an assumption and other types of artifacts. As an example, the content of a decision focuses on giving a solution to a problem, while the content of an assumption deals with whether something is correct, suitable, valuable, etc. [2].

On the Manually Labeling Posts step, we randomly selected 904 sentences from the 843 posts that contain assumptions. To mitigate any bias in getting the truth set, we first conducted a pilot labeling with 200 sentences from the 904 sentences by the first and sixth authors independently. Conflicts were discussed and resolved with the second author, in order to make sure that they had a consistent understanding about the criteria of data labeling. The formal data labeling was then conducted in two steps: (1) The first and sixth authors labeled the remaining 704 sentences independently and we measured the inter-rater reliability and calculated Cohen's Kappa coefficient [32] for labeling sentences between the two authors and obtained an agreement of 0.821. Similar to the pilot labeling, disagreements on the labeled sentences were discussed and resolved with the second author. (2) Finally, we got 468 "*Assumption*" sentences and 436 "*Non-Assumption*" sentences from the 904 sentences. All the data of this study has been provided online[6].

We randomly selected 400 "*Assumption*" sentences and 400 "*Non-Assumption*" sentences from the 904 sentences to form the dataset. The dataset was then split into two parts: (a) 75% of the sentences as the training set, and (b) 25% of the sentences as the testing set.

[5] https://sourceforge.net/projects/hibernate/
[6] https://tinyurl.com/y4tr7v78

### B. Classifiers Selection

We selected seven popular supervised learning methods in the field of machine learning, including Perceptron (Pct), Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Classification And Regression Tree (CART), Naive Bayes (NB), and Support Vector Machines (SVM). These algorithms are also used in similar studies, e.g., the classification of non-functional requirements [33] and rationale [34]. We utilized the seven classification algorithms to train seven classifiers for automatically identifying assumptions.

### C. Evaluation Metrics

We evaluate the classifiers through precision and recall, both of which are two widely used metrics in measuring the relevance. Precision is usually used to calculate the correctly classified assumptions related sentences in respect to the total number of sentences retrieved, while recall is usually used to calculate the proportion of relevant sentences that are successfully retrieved. Formula (1) and (2) define the calculation of precision and recall respectively. To balance precision and recall as a harmonic mean, we measure the classification performance through $F_1$-score based on the precision and recall results (using ten-fold cross-validation). $F_1$-score is widely used for information retrieval tasks and corresponds to the trade-off value of precision and recall [35], as shown in Formula (3).

$$precision = \frac{|relevant\_Sentences \cap retrived\_Sentences|}{|retrived\_Sentences|} \quad (1)$$

$$recall = \frac{|relevant\_Sentences \cap retrived\_Sentences|}{|relevant\_Sentences|} \quad (2)$$

$$F_1 - score = \frac{2 \times precision \times recall}{precision + recall} \quad (3)$$

Furthermore, to eliminate the potential and undesired bias from $F_1$-score [36], we compared the holistic performance of the seven classifiers by plotting the ROC curves and calculating the AUC values. The results of the ROC curves and the AUC values can be found in Section V.

## V. Results and Discussion

In this section, we present and analyze the experiment results, and discuss the performance of the seven classification algorithms for automatic assumption identification through a labeled dataset.

### A. Results and Discussion of RQ1

TABLE III presents the classification results of the seven classifiers. Overall, the precision, recall, and $F_1$-score values (via ten-fold cross-validation) of the seven supervised ML methods show a satisfactory result. To be more specific, we can accurately identify assumptions from textual information through training classifiers, as the $F_1$-scores of six out of the seven classifiers exceed 0.7.

Moreover, the results show that (1) the standard deviation (SD) of each classifier's precision, recall, and $F_1$-score is less than 0.1, which means that the seven trained classifiers are

appropriate to identify assumptions; (2) the precision, recall, and $F_1$-score of the classifiers based on Pct, LR, LDA, NB, and SVM-based algorithms are comparatively stable than the KNN-based and CART-based classifier. Thus those five classifiers can get more reliable $F_1$-score. Overall, these results show that the trained classifiers can be applied to automatically identify assumptions with acceptable performance.

TABLE III. RESULTS OF THE SEVEN CLASSIFICATION ALGORITHMS

| # | Precision | | Recall | | $F_1$-score | |
|---|---|---|---|---|---|---|
| | Mean | SD | Mean | SD | Mean | SD |
| Pct | 0.767 | 0.075 | 0.768 | 0.051 | 0.765 | 0.049 |
| LR | 0.788 | 0.071 | 0.783 | 0.050 | 0.783 | 0.046 |
| LDA | 0.785 | 0.062 | 0.798 | 0.045 | 0.789 | 0.035 |
| KNN | 0.847 | 0.059 | 0.662 | 0.064 | 0.740 | 0.042 |
| CART | 0.691 | 0.072 | 0.636 | 0.090 | 0.664 | 0.076 |
| NB | 0.783 | 0.050 | 0.773 | 0.069 | 0.776 | 0.052 |
| SVM | **0.829** | 0.059 | **0.812** | 0.071 | **0.819** | 0.055 |

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

We compared the results to evaluate the performance of the seven classifiers, in order to find out the best classifier for automatic identification of assumptions. Figure 2 shows distinctions of the seven classifiers, i.e., the mean values calculated via ten-fold cross-validation. The SVM-based classifier gets the highest precision (0.829), recall (0.812), and $F_1$-score (0.819), followed by the LDA-based classifier, with precision (0.785), recall (0.798), and $F_1$-score (0.789).

In addition, in order to provide a holistic and more cogent result, as well as mitigate the bias potentially from $F_1$-score [36] (because $F_1$-score may incline to positive class, especially for imbalanced dataset [36]), we made a further comparison for the performance of the seven classifiers by calculating the area under the Receiver Operating Characteristic (ROC) curve [37], which is also known as Area Under the Curve (AUC).

ROC curve can reflect how well a classification model performs and it is a comparatively straightforward way to judge the performance of a classifier. The AUC value is a measurement of the area under the ROC curve. The AUC value of a classifier with a 100% wrong prediction is 0.0; while the AUC value of a classifier with a 100% correct prediction is 1.0 [38]. Figure 3 shows the ROC curves of the seven classifiers, and the corresponding AUC values were calculated and shown in the legend. True Positive Rate (TPR) and False Positive Rate (FPR) are two opposite metrics in Figure 3, which are also called Sensitivity and 1-Specificity (calculated by Formula (4) and (5) respectively). True Positive (TP) represents the number of correctly classified assumptions, False Positive (FP) represents the number of incorrectly classified assumptions, True Negative (TN) represents the number of assumptions correctly not classified, and False Negative (FN) is the number of assumptions incorrectly not classified.

$$TPR = Sensitivity = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = 1 - Specificity = \frac{FP}{FP + TN} \tag{5}$$

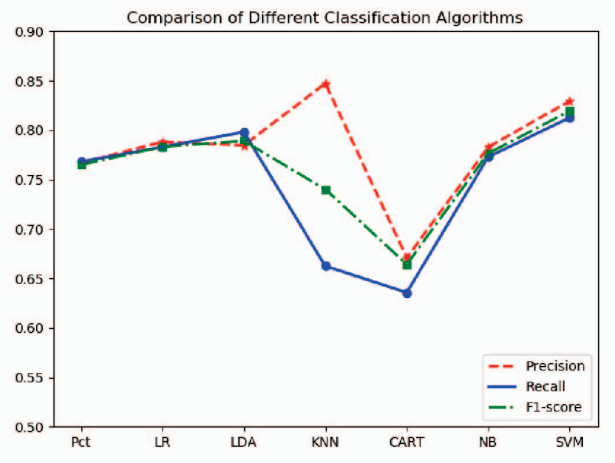


Fig. 2 Comparison of the seven ML algorithms for assumption classification

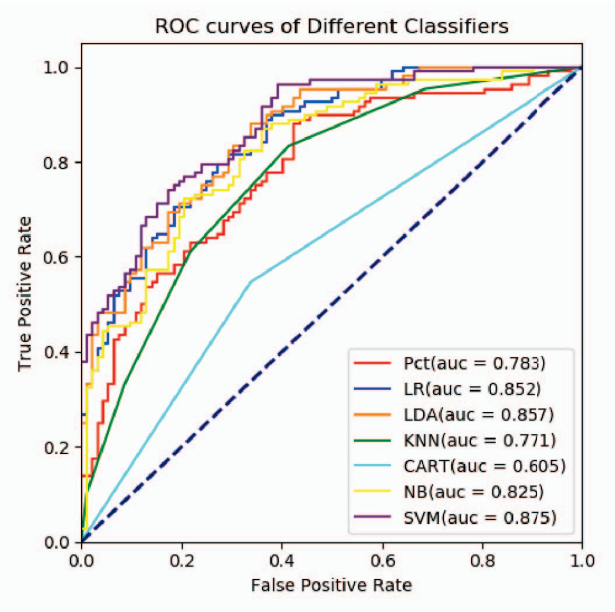


Fig. 3 ROC curves and AUC values of the seven classifiers

An AUC value of 0.5 (see the blue dashed line in Figure 3) is similar to a random selection, and we selected 0.5 as the threshold to compare the performance of classifiers (i.e., the more margin over threshold, the better performance). According the results in Figure 3, the SVM-based classifier achieves the highest AUC value of 0.875, rendering it the best classifier in the experiment. The LR-based and LDA-based classifiers are ranked in the second position with a similar performance, which is consistent to the results of RQ1. In conclusion, compared to the other six classifiers, the SVM-based classifier has the best performance for automatic identification of assumptions.

## VI. IMPLICATIONS

### A. Designing Tools for Automatic Identification of Assumptions

Considering that the SVM-based classifier has the best performance on the automatic identification of assumptions, it is important to integrate it into relevant tools for practical use by developers. In addition, we advocate that both researchers and practitioners should participate in the design of automatic tools for identifying assumptions in software development, which can alleviate early deficiencies at the design level of those supporting tools.

### B. Establishing Shared Dataset related to Assumptions

Assumptions, as a type of inherent artifact in software development, should get more attention from especially practitioners. Moreover, the lack of widely adopted labeled datasets of assumptions is one of the main obstacles for the application of automated techniques (e.g., training ML-based classifiers needs large data). Therefore, collaborations between the software engineering and the machine learning community would be valuable in order to establish a labeled common dataset of assumptions. Such dataset can facilitate further research as well as practice on assumptions and their management. We also believe that a widely accepted corpus and dataset can pave the way towards extensive applications of automated techniques.

## VII. THREATS TO VALIDITY

There are several threats to the validity of this study. We discuss the threats according to the guidelines in [39]. Internal validity was not considered, since we did not address causal relationships between variable and results.

### A. Construct Validity

Construct validity focuses on whether the theoretical or conceptual constructs are interpreted and measured correctly. A potential threat is if the dataset is accurate with respect to the labeled data used in this study. For instance, different researchers may have different understandings regarding the textual information. To mitigate this threat, we reviewed and labeled the textual information by two researchers independently and another researcher revised the results when there were disagreements. In addition, we conducted a pilot study through selecting 200 sentences and labeled by two researchers independently. Any conflicting classifications were further discussed and resolved to eliminate personal bias.

### B. External Validity

External validity concerns the extent to which our findings from this study can be generalized in other settings. As discussed in [30], we consider Hibernate as a representative OSS project to a large extent in OSS development. Therefore, the method and results of our work can be generalized to other classification problems in a similar context. Meanwhile, we believe that our method (see Section III) can be adapted and employed for automatic identification of other types of textual artifacts (e.g., design decisions).

### C. Reliability

Reliability refers to the consistency of a measurement. In other words, it is related to whether the experiment is conducted and present in such a way so that when other researchers replicate it will reach the same results. To reduce this threat, the research protocol was discussed and confirmed by all the researchers iteratively, and all the details of the study are provided in Section III with the data of this study available online (see Section IV.A). We believe that this threat has been partially mitigated.

## VIII. CONCLUSIONS AND FUTURE WORK

Since developers from all over the world can contribute to an OSS project, the communication with the other developers is not easy due to the geographical distance. One of the most common ways that the developers use to communicate is through mailing lists. Therefore, those mailing lists usually have useful knowledge about the project context as well as decisions and assumptions made during the development. Automatic identification of assumptions from developer mailing lists can help to automatically extract that knowledge which is normally implicit in OSS development, and this can be useful for existing developers to better understand the project, and especially for the developers that are new to the project.

In the software development life cycle, assumptions are an important type of software development information that can be extracted from textual artifacts. In software design, analyzing assumptions can help to comprehend the underlying design decisions during the development phase and further facilitate maintenance of software systems. Manual identification of assumptions by stakeholders is rather time-consuming, especially when analyzing a large amount of textual artifacts. To address this problem, one promising way is to use automatic techniques for assumption identification, which can significantly alleviate the effort of stakeholders.

In this paper, we established a dataset extracted from the Hibernate developer mailing list, and this dataset is composed of 400 "*Assumption*" sentences and 400 "*Non-Assumption*" sentences. We then used the dataset in the experiments for training and evaluating seven classifiers based on seven ML classification algorithms. The results show that the SVM algorithm achieved the best performance (with a precision of 0.829, a recall of 0.812, and an $F_1$-score of 0.819). According to the ROC curves and related AUC values, the SVM-based classifier comparatively performs better than the other classifiers in the binary classification of assumptions.

Given the importance of assumptions in software development, our future work is to provide a multiple classifier model to realize automatic identification of various types of assumptions using different textual artifacts, which would be more effective in managing various types of assumptions in the software development life cycle. Furthermore, we also plan to employ the classifiers with more OSS and industrial projects for evaluating the performance of the classifiers in a real-world setting.

## ACKNOWLEDGMENTS

This work is partially sponsored by the National Key R&D Program of China with Grant No. 2018YFB1402800. The authors gratefully acknowledge the financial support from the China Scholarship Council.

## REFERENCES

[1] C. Yang, P. Liang, P. Avgeriou, U. Eliasson, R. Heldal, and P. Pelliccione, "Architectural assumptions and their management in industry–An exploratory study," in Proceedings of the 11th European Conference on Software Architecture (ECSA), 2017, pp. 191-207: Springer.

[2] C. Yang, P. Liang, and P. Avgeriou, "Assumptions and their management in software development: A systematic mapping study," Information and Software Technology, vol. 94, pp. 82-110, 2018.

[3] Ö. Albayrak, H. Kurtoglu, and M. Biçakçi, "Incomplete software requirements and assumptions made by software engineers," in Proceedings of the 16th Asia-Pacific Software Engineering Conference (APSEC), 2009, pp. 333-339: IEEE.

[4] R. Ali, F. Dalpiaz, P. Giorgini, and V. E. S. Souza, "Requirements evolution: from assumptions to reality," in Enterprise, Business-Process and Information Systems Modeling: Springer, 2011, pp. 372-382.

[5] T. Arts, M. Dorigatti, and S. Tonetta, "Making implicit safety requirements explicit," in Proceedings of the 33th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), 2014, pp. 81-92: Springer.

[6] A. Cimatti and S. Tonetta, "A property-based proof system for contract-based design," in Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2012, pp. 21-28: IEEE.

[7] S. Faily and I. Fléchais, "The secret lives of assumptions: Developing and refining assumption personas for secure system design," in Proceedings of the 3rd International Conference on Human-Centred Software Engineering (HCSE), 2010, pp. 111-118: Springer.

[8] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2010, pp. 23-37: Springer.

[9] M. M. Lehman, "The role and impact of assumptions in software development, maintenance and evolution," in Proceedings of the 1st IEEE International Workshop on Software Evolvability (IWSE), 2005, pp. 3-14: IEEE.

[10] N. H. Pham, V.-H. Nguyen, T. Aoki, and T. Katayama, "An improvement of minimized assumption generation method for component-based software verification," in Proceedings of the International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012, pp. 1-6: IEEE.

[11] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is still so hard," IEEE Software, vol. 26, no. 4, pp. 66-69, 2009.

[12] A. Steingruebl and G. Peterson, "Software assumptions lead to preventable errors," IEEE Security & Privacy, vol. 7, no. 4, pp. 84-87, 2009.

[13] M. M. Lehman and J. F. Ramil, "Rules and tools for software evolution planning and management," Annals of software engineering, vol. 11, no. 1, pp. 15-44, 2001.

[14] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 277-286: IEEE Press.

[15] G. A. Lewis, T. Mahatham, and L. Wrage, "Assumptions management in software development," Technical Report, Carnegie Mellon University2004.

[16] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," Journal of Systems and Software, vol. 80, no. 6, pp. 918-934, 2007.

[17] M. A. A. Mamun and J. Hansson, "Review and challenges of assumptions in software development," in Proceedings of the 2nd Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS), 2011.

[18] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools," in Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture and 3rd European Conference on Software Architecture (WICSA/ECSA), 2009, pp. 293-296: IEEE.

[19] K. Shumaiev and M. Bhat, "Automatic Uncertainty Detection in Software Architecture Documentation," in Proceedings of the International Conference on Software Architecture Workshops (ICSAW), 2017, pp. 216-219: IEEE.

[20] P. Velasco-Elizondo, R. Marín-Piña, S. Vazquez-Reyes, A. Mora-Soto, and J. Mejia, "Knowledge representation and information extraction for analysing architectural patterns," Science of Computer Programming, vol. 121, pp. 176-189, 2016.

[21] L. Pascarella and A. Bacchelli, "Classifying code comments in Java open-source software systems," in Proceedings of the 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 227-237: IEEE.

[22] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," Requirements Engineering, vol. 21, no. 3, pp. 311-331, 2016.

[23] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric Approach," Encyclopedia of Software Engineering, pp. 528-532, 1994.

[24] F. Shull, J. Singer, and D. I. Sjøberg, Guide to advanced empirical software engineering. Springer, 2007.

[25] S. Bird, E. Klein, and E. Loper, Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009.

[26] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," Journal of machine learning research, vol. 12, no. Oct, pp. 2825-2830, 2011.

[27] R. Visser and B. O. K. Intelligentie, "Tag cloud visualisation of verbal discussions following speech-to-text," 2015.

[28] X. Rong, "word2vec parameter learning explained," arXiv preprint arXiv:1411.2738, 2014.

[29] C. Yang, P. Liang, and P. Avgeriou, "Evaluation of a process for architectural assumption management in software development," Science of Computer Programming, vol. 168, pp. 38-70, 2018.

[30] Z. Xiong, P. Liang, C. Yang, and T. Liu, "Assumptions in OSS Development: An Exploratory Study through the Hibernate Developer Mailing List," in Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC), 2018, pp. 455–464: IEEE.

[31] H. Ziv, D. Richardson, and R. Klösch, "The uncertainty principle in software engineering," in Proceedings of the submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97), 1997.

[32] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," biometrics, pp. 159-174, 1977.

[33] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in Proceedings of the 25th IEEE International Requirements Engineering Conference (RE), 2017, pp. 496-501: IEEE.

[34] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge, "How do developers discuss rationale?," in Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018, pp. 357-369: IEEE.

[35] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling, "Feature selection using linear classifier weights: interaction with classification models," in Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2004, pp. 234-241: ACM.

[36] G. Forman and M. Scholz, "Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement," ACM SIGKDD Explorations Newsletter, vol. 12, no. 1, pp. 49-57, 2010.

[37] T. Fawcett, "An introduction to ROC analysis," Pattern recognition letters, vol. 27, no. 8, pp. 861-874, 2006.

[38] E. Alpaydin, Introduction to machine learning. MIT press, 2009.

[39] M. B. Brewer and W. D. Crano, "Research design and issues of validity," Handbook of Research Methods in Social and Personality Psychology, pp. 3-16, 2000.