Content Categorization of API Discussions

Daqing Hou
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, New York 13699
Email: dhou@clarkson.edu

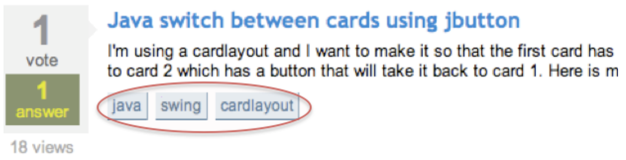
Lingfeng Mo
Department of Computer Science
Clarkson University
Potsdam, New York 13699
Email: mol@clarkson.edu

***Abstract*—*Text categorization*, automatically labeling natural language text with pre-defined semantic categories, is an essential task for managing the abundant online data. An example of such data in Software Engineering is the large, ever-growing volume of forum discussions on how to use particular APIs. We have conducted a study to explore the question as to how well machine learning algorithms can be applied to categorize API discussions based on their content. Our goal is two-fold: (1) Can a relatively straightforward algorithm such as Naive Bayes work sufficiently well for this task? (2) If yes, how can we control its performance? We have achieved the best test accuracy mean (TAM) of 94.1% with our largest training data set for the AWT/Swing API, which consists of 833 forum discussions distributed over eight categories/topics. We have also investigated factors that impact classification accuracy, with the most important two being the size of the training set and multi-label documents (the phenomenon that some discussions involve more than one category).**

***Index Terms*—APIs; Online Forums; Text Categorization; Naive Bayes; MALLET; AWT/Swing.**

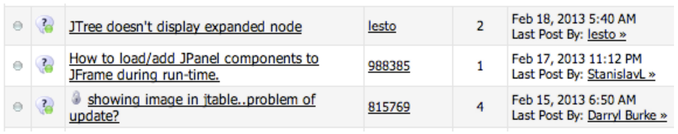
I. Introduction

Online forums are commonly used by software developers to exchange information, asking questions and seeking help. Over time, these forums have become repositories of valuable programming tips and knowledge that many developers have learned to revisit routinely. To facilitate the effective browsing and searching of software forums, however, we need to find additional ways to index the forum data and better reveal their semantics, such as the Stack Overflow tags [1] shown below.


1 vote
1 answer
18 views
Java switch between cards using jbutton
I'm using a cardlayout and I want to make it so that the first card has to card 2 which has a button that will take it back to card 1. Here is m
java swing cardlayout

However, Stack Overflow relies on users to manually assign tags. *Text categorization*, automatically labeling natural language texts with pre-defined categories based on their content, can be applied to automate this task [20].

In this paper, we investigate *text categorization* as a fundamental approach to learn the semantic information needed for tagging forum data with pre-defined categories. This additional piece of information can then be used, for example, to annotate the master list of discussion threads in the Swing Forum that is shown below, to make it more explicit and informative as to what each entry is about.



JTree doesn't display expanded node	lesto	2	Feb 18, 2013 5:40 AM Last Post By: lesto »
How to load/add JPanel components to JFrame during run-time.	988385	1	Feb 17, 2013 11:12 PM Last Post By: StanislavL »
showing image in jtable..problem of update?	815769	4	Feb 15, 2013 6:50 AM Last Post By: Darryl Burke »

Our goal has been two-fold: First, we want to investigate the feasibility of using existing machine learning algorithms to categorize forum discussions. Second, in order to guide similar efforts in the future, we want to gain insights as to exactly how a learning algorithm works for this task.

We chose to start with the Naive Bayes algorithm [13] to categorize the API discussions in the Swing Forum [1]. Historically, Naive Bayes has been found performing “remarkably well” for many tasks [12], [14]. Although more recent studies have compared it with other algorithms [22], [10], due to the empirical nature of such comparisons, the results may not be universally true. Moreover, if Naive Bayes works reasonably well for our task, other more “superior” algorithms should only perform even better, e.g., the SVM comparison in Section IV-J.

We chose the Swing Forum because it is a very active forum; as of February 12, 2013, it contains 47,258 discussion threads with 212,734 messages. We treat a whole discussion thread, which typically contains multiple messages/posts, as a training document. So it is crucial that we assign a label that accurately reflects the content of each discussion thread. An advantage of choosing the Swing Forum is that we can leverage the considerable prior research that we have conducted with the forum, e.g., [9], [19], so we can have greater confidence with the quality of our training data.

The contributions of this work are summarized as follows:

- Publicly available training data sets: To experiment with Naive Bayes, we manually labeled three sets, a total of approximately 1,000 API discussions collected from the Swing Forum. Our largest training data set consists of 833 API discussions across eight categories specific to Java Swing. We have made our data publicly available [2].
- Using our data, we show that the Naive Bayes classifier can achieve an average test accuracy as high as 94.1%.
- We demonstrate empirically that the training sample size is the most important factor for improving classification accuracy, but this increase of accuracy plateaus once the training sample size surpasses a certain threshold.

[1]https://forums.oracle.com/forums/forum.jspa?forumID=950&start=0 (All URLs verified Feb. 12, 2013)
[2]**http://www.clarkson.edu/~dhou/projects/swingForum2012.tar.gz**

- We demonstrate that multi-label documents are the major cause for classification errors.

The remainder of this paper is organized as follows. Section II describes related work. Section III covers the background for the Naive Bayes algorithm. Section IV presents our experimentation process and main results. Finally, Section V concludes the paper.

II. Related Work

We survey some applications of machine learning to classification problems in Software Engineering. In general, different from related work, our study uses semantics-oriented classification categories. Moreover, we not only show that the algorithm works, but also investigate why and how it works.

A. Software Forums and Email Communication

Gottipati, Lo, and Jiang present an approach where tags automatically inferred for posts in software forum threads are used to find relevant answers in the forums [7]. The tag inference algorithm automatically assigns seven different tags to posts: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk. Experiments show that their tag inference could achieve up to 67% precision, 71% recall, and 69% F-measure. They also build a semantic search engine by leveraging the inferred tags to find relevant answers. Unlike ours, the unit of classification in their work is a message/post, rather than a whole discussion. Furthermore, their tags are about the conversational role that a message/post plays in a discussion thread, rather than the API-specific semantics of the whole discussion.

Bacchelli et al. present an approach to classify email lines in five categories (i.e., text, junk, code, patch, and stack trace) so that one can subsequently apply appropriate ad hoc analysis techniques to each category [5].

In this study, we have expanded two sets of Swing Forum discussions that we collected and analyzed as part of two of our own prior research projects [9], [19]. The first set consists of 172 discussions, which were analyzed to understand the kinds of API obstacles programmers may have in practice [9]. The second set consists of 117 discussions about the GUI layout issues, which were analyzed to drive the development of a program analysis tool supporting the use of APIs [19]. These two sets are included in our final data set of 833 discussions.

B. Bug Repositories

Open source development projects typically support an open bug repository to which both developers and users can report bugs. The reports that appear in this repository must be triaged to determine if the report is one which requires attention and if it is, which developer will be assigned the responsibility of resolving the report. Large open source developments are burdened by the rate at which new bug reports appear in the bug repository. To ease the task of assigning reports to a developer, Anvik, Hiew, and Murphy [4] present a semi-automated approach where they apply a machine learning algorithm to the open bug repository to learn the kinds of reports each developer resolves. When a new report arrives, the classifier produced by the machine learning technique suggests a small number of developers suitable to resolve the report. They have reached precision levels of 57% and 64% on the Eclipse and Firefox development projects respectively, as well as less positive results for the gcc open source development.

A related problem in open bug repositories is duplicate bug reports that require triaging. To help with the triage process, Sun et al. [21] present a machine learning approach that can be used to classify whether a new bug report is a duplicate of any existing ones. Their techniques have been validated on three large software bug repositories from Firefox, Eclipse, and OpenOffice, showing 17–31%, 22–26%, and 35–43% relative improvement over state-of-the-art techniques.

Antoniol et al. [3] study how machine learning algorithms can be used to separate true bug reports from other change requests. They find that alternating decision trees, Naive Bayes classifiers, and logistic regression can be used to accurately distinguish bugs from other kinds of issues. Results from empirical studies performed on issues for Mozilla, Eclipse, and JBoss indicate that issues can be classified with between 77% and 82% of correct decisions.

C. Change Classification

Large software systems undergo significant evolution during their lifespan, yet often individual changes are not well documented. Hindle et al. [8] present a study to automatically classify large changes into various categories of maintenance tasks - corrective, adaptive, perfective, feature addition, and non-functional improvement - using machine learning techniques. They find that for most large commits, the Source Control System (SCS) commit messages plus the commit author identity are enough to accurately and automatically categorize the nature of the maintenance task. Using 10-fold cross validation, they achieved accuracies consistently above 50%, indicating good to fair results.

Kim et al. [11] study a new technique for finding latent software bugs called change classification. Change classification uses a machine learning classifier to determine whether a new software change is more similar to prior buggy changes, or clean changes. In this manner, change classification predicts the existence of bugs in software changes. The classifier is trained using features (in the machine learning sense) extracted from the revision history of a software project, as stored in its software configuration management repository. The trained classifier can classify changes as buggy or clean with 78% accuracy and 65% buggy change recall (on average). Murgia et al. [17] present another study with similar goals.

III. The Naive Bayes Algorithm

Text categorization involves the classification of text documents into a set of categories [20]. When classifying API discussions, the text documents are the discussion threads, and the categories are the central topics that are discussed in the threads. In machine learning, documents are called instances and attributes of an instance are called features. Instances may also have a label that indicates the category, or class, to which it belongs. A supervised machine learning algorithm takes as input a set of instances with known labels and generates a classifier, which can then be used to assign a label to an unknown instance. The process of creating a classifier from a set of instances is known as *training the classifier*.

Our work focuses on using supervised learning for classifying API discussions. We used the MALLET machine learning toolkit for language processing [15]. In what follows, we briefly review the Naive Bayes algorithm, about which more details can be found elsewhere [13], [16], [18], [12].

Naive Bayes learns a classifier from a training sample that is made of multiple training instances, where each instance is labeled with one of k pre-defined categories $C_1, C_2, \ldots, C_k$. Given a textual document D with an unknown category, the learned Naive Bayes classifier assigns D a category C, which is the category C_i that yields the highest conditional probability $P(C_i \mid D)$. Formally, C is chosen according to Equation 1:

$$C = arg_{C_i} max P(C_i \mid D) \tag{1}$$

Therefore, to choose C, it is necessary to learn to calculate $P(C_i \mid D)$. Based on the Bayes Theorem shown in Equation 2,

$$P(A \mid B) * P(B) = P(B \mid A) * P(A) \tag{2}$$

$P(C_i \mid D)$ can be calculated according to Equation 3:

$$P(C_i \mid D) = \frac{P(D \mid C_i) * P(C_i)}{P(D)} \tag{3}$$

Among the three terms in the right-hand side of Equation 3, the denominator $P(D)$ is effectively constant, and $P(C_i)$ can be calculated as the ratio between the number of training instances in category C_i and the total number of training instances in the training sample. Therefore, for the purpose of choosing C according to Equation 1, indeed it is only necessary to learn to calculate $P(D \mid C_i)$, and this is where the "Naive" assumption comes into play.

Given a vocabulary of n terms (features) $T_1, T_2, \ldots, T_n$ selected from the training data, Naive Bayes assumes that these terms be independent of context and position. Based on this independence assumption, $P(D \mid C_i)$ can be approximated by $\bar{P}(D \mid C_i)$, which is calculated as the product of $P(T_j \mid C_i)$, the probability of term T_j appearing in category C_i:

$$\bar{P}(D \mid C_i) = \prod_{1 \le j \le n} P(T_j \mid C_i)^{\#(T_j, D)} \tag{4}$$

where $\#(T_j, D)$ represents the frequency by which term T_j appears in document D. This formulation is essentially the multinomial model introduced in [16], [13] but with some combinatorial coefficients removed for simplification. Although the Naive Bayes assumption does not hold for natural languages in general, in practice, this algorithm has been shown to work sufficiently well for many applications [12].

$P(T_j \mid C_i)$ is in turn approximated by the sample probability $\bar{P}(T_j \mid C_i)$, which is calculated according to Equation 5,

$$\bar{P}(T_j \mid C_i) = \frac{1 + \sum_k \#(T_j, D_{i,k})}{n + \sum_k \#(D_{i,k})} \tag{5}$$

where k ranges over all training documents belonging to category C_i, $\#(T_j, D_{i,k})$ the frequency by which term T_j appears in document $D_{i,k}$, n the vocabulary size, and $\#(D_{i,k})$ the total number of term occurrences in document $D_{i,k}$.

Due to the product of many terms [3] in Equation 4, underflow may occur. To prevent underflow, the actual implementation of Naive Bayes in MALLET [15] uses the logarithm of $\bar{P}(D \mid C_i)$ instead. Therefore, $\bar{P}(T_j \mid C_i)$ cannot be zero. The **1** in Equation 5 is added to ensure that $\bar{P}(T_j \mid C_i)$ is never zero.

MALLET's implementation of Naive Bayes uses the multinomial model, which has been shown to work better than the multi-variate Bernoulli model [16] for categorization tasks that have varying document lengths and large vocabularies.

IV. Experimental Process and Results

In this study, we want to explore two research questions:

RQ1 Will existing machine learning algorithms in general and Naive Bayes in particular perform "sufficiently well" for categorizing API discussions? Since initially we did not know what to expect, we considered that a rather modest average test accuracy of 70% or above would be "sufficiently well" for practical use in Software Engineering.

RQ2 If the answer to the first question is yes, we would like to further explore the major factors that impact the classification accuracy.

Since our goal is not to advance the state-of-art in machine learning, but to investigate the feasibility of using existing algorithms to solve particular software engineering problems, we chose to start with the Naive Bayes algorithm [13]. This is because historically Naive Bayes has been found performing "remarkably well" for many tasks [12], [14]. Although more recent studies have compared it with other algorithms [22], [10], because of the empirical nature of such comparisons, the results may not be universally true. Finally, if Naive Bayes works reasonably well for our task, other more "superior" algorithms should only perform even better.

We chose to categorize the API discussions from the Swing Forum because it is a very active forum. Another advantage is that we can leverage the considerable prior research that we have conducted with the forum, e.g., [9], [19], so we can have greater confidence with the quality of our training data. We treat a whole discussion thread, which typically has multiple messages/posts, as a training document.

A. Data Collection

The most time-consuming task in text categorization is preparing training data [18]. In our case, unlike other studies [3], [8], the categories are domain-specific and do not exist beforehand. Therefore, we need to come up with both

[3]Even with stop words removed, our Version 3 training data produces a vocabulary of 9,144 words!

the categories and the training documents for each category. Moreover, to ensure data authenticity, we must clearly define each category. In addition, we were not sure about what would be the minimum number of documents for each category in order to achieve reliable, good-enough learning. Consequently, we have evolved three versions of single-labeled training data. Table I depicts the three versions of data, and Table II summarizes the classification accuracy for all of our experiments.

Version 1 was created quickly to test the Naive Bayes algorithm. It contains ten categories and 46 discussion threads, with the minimum and maximum numbers of documents for each category being 3 and 10, respectively. As seen from Table II, the average test accuracy for Version 1 is low.

Version 2 increased the training sample size to 158 documents and 17 categories, with the minimum and maximum numbers of documents for each category being 6 and 13, respectively. Our goal was to investigate the effect of sample size on classification accuracy. Although the average test accuracy for Version 2 is not adequate either, the improvements in comparison with Version 1 are noticeable. Notice that as a result of refining and improving the definitions of the ten categories in Version 1, we have added seven new categories to Version 2, which are included for the sake of completeness.

In order to further confirm the effect of increased sample size on classification accuracy, we populated the "layout" category with 117 layout-related discussions that we have analyzed thoroughly in a prior study [19]. Because the result showed clearly the effectiveness of sample size for improving classification accuracy, we selected eight categories from Version 2 and aggressively increased the size of each of them to about 100 documents to form Version 3. Using Version 3, we were able to answer **RQ1** satisfactorily with an above-90% average classification accuracy and to investigate **RQ2**.

In general, in order to have sound judgments about label assignment in training data preparation, the experimenter must

- Possess sound knowledge for the subject matter to be categorized.
- Fully understand the content of a discussion. It is critical for ensuring the quality of our study and accurately assigning labels for documents. As a result, we carefully read through the discussions to understand the details contained in the collected documents.
- Understand that code quoted in a discussion may complement verbal descriptions. This is because the discussion participants may not always be able to accurately describe their problems, especially in the case of an API novice.
- Avoid the tendency to assign a category based on inferred information that is not described literally in the discussion, such as the fact that there is a certain bug.

B. MALLET Commands and Evaluation Metrics

We used the MALLET machine learning toolkit for language processing [15]. Training data are organized as two-level file folders. Documents belonging to the same category are put under the same folder. The folders for all categories are in turn put under a top-level folder.

TABLE I: Three versions of training data collected from Java Swing Forum and their breakdown by categories

Data Version	Categories/Labels	#Documents	Total
V1.0	border	4	46
	dispose	3	
	drawing	3	
	focus	3	
	layout	8	
	action	10	
	icon	5	
	rendererEditor	4	
	title	3	
	others	3	
V2.0	borderAndMargin	13	158
	dispose	9	
	drawing	8	
	focus	9	
	layout	9	
	action	8	
	loadingIcons	8	
	rendererEditor	13	
	titleBar	6	
	titleBarFont	10	
	textIconPosition	10	
	dynamicHierarchy	10	
	defaultButton	11	
	mouseMotionPosition	12	
	threading	8	
	OOP	7	
	social	7	
V3.0	borderAndMargin	82	833
	dispose	103	
	drawing	108	
	focus	104	
	layout	125	
	titleBar	106	
	textIconPosition	96	
	dynamicHierarchy	109	

As the first step, MALLET imports and converts the training data into the vector space format that can be used for machine learning, indicating the frequencies of the terms in the text, with the following command:

```
% mallet import-dir --input v10/*
--output v10.mallet
```

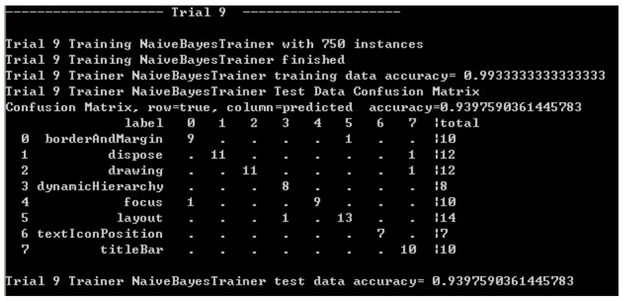
This example converts Version 1 text stored in `v10` into a file named "`v10.mallet`" to be used for machine learning.

The next step is to train a classifier with the data file created above, with the following command:

```
% mallet train-classifier --input
v10.mallet --training-portion 0.9
--num-trials 10000
```

Recall that when training a classifier with a machine learning algorithm, the training instances with pre-classified categories are split into two sets, the *training (-and-validation) set* and the *test set*, to be used for training and testing the classifier, respectively [20]. According to the `training-portion` option for the command above, MALLET will randomly split the training data into 90% training (-and-validating) instances for learning the classifier, and 10% for testing it. Furthermore, according to the `num-trials` option, this training-and-testing process will be repeated 10,000 times, each of which is called a trial.

The output for each trial includes a confusion matrix, each



```
------------------- Trial 9  --------------------

Trial 9 Training NaiveBayesTrainer with 750 instances
Trial 9 Training NaiveBayesTrainer finished
Trial 9 Trainer NaiveBayesTrainer training data accuracy= 0.9933333333333333
Trial 9 Trainer NaiveBayesTrainer Test Data Confusion Matrix
Confusion Matrix, row=true, column=predicted  accuracy=0.9397590361445783
              label   0   1   2   3   4   5   6   7  |total
  0  borderAndMargin   9   .   .   .   .   1   .   .  |10
  1          dispose   .  11   .   .   .   .   .   1  |12
  2          drawing   .   .  11   .   .   .   .   1  |12
  3 dynamicHierarchy   .   .   .   8   .   .   .   .  |8
  4            focus   1   .   .   .   9   .   .   .  |10
  5           layout   .   .   .   1   .  13   .   .  |14
  6 textIconPosition   .   .   .   .   .   .   7   .  |7
  7         titleBar   .   .   .   .   .   .   .  10  |10

Trial 9 Trainer NaiveBayesTrainer test data accuracy= 0.9397590361445783
```

Fig. 1: Sample confusion matrix as a result of one trial with Version 3 (Training set of 750 documents; 83 test documents; 5 of the 83 test documents incorrectly classified -the off-diagonal cells-, resulting in a test accuracy of 93.98%.)

row of which represents the classification results for one category and whose cells show how the learned classifier labels the test instances of that category. Figure 1 depicts an example of a confusion matrix. For each row, the diagonal element shows the number of correct predictions, and the off-diagonal elements show the numbers of classification errors.

The test accuracy is defined as the ratio between the number of correctly labeled instances and the total number of test instances. After all trials are done, an overall measure of average test accuracy, TAM (Test Accuracy Mean), is calculated, as a metric for the accuracy of the learned classifier.

C. Feature Selection

We have applied three feature selection methods, whose impact on classification accuracy is summarized by Table II.

1) Stop Words Removal (SWR): Common, generic function words, such as "a", "the", "how", "which", "what", and "where", which are also known as stop words, do not contribute to the discrimination power needed for text categorization and, thus, can be removed from the vocabulary. MALLET provides two options to remove stop words from the input text during the importing process. The first option instructs MALLET to ignore 524 common English adverbs, conjunctions, pronouns, and prepositions, a list of standard stop words that MALLET defines as default. The second option allows for adding extra stop words beyond the default.

The vocabulary size for Version 3 is 9,617, and 9,144 after stop words removal.

2) Words Splitting (WS): Source code is commonly quoted in API discussions. The lexical identifiers from the quoted source code usually contain useful information for defining the main topics of a discussion, which may complement the verbal description in important ways. For example, the appearance of a type name such as `JActionListener`, or a method call such as `addActionListener(aListener)` in a discussion, indicates that the discussion is at least partially about action handling.

However, due to the adoption of popular naming conventions such as Camel Cases and Underscoring, the lexical identifiers from source code cannot be used effectively for

TABLE II: Test Accuracy Mean (TAM) and the associated stddev for all the training-and-testing experiments with the three versions of training data, all in percentage. Best accuracy results for each version are highlighted in bold. The following feature selection methods are used: RAW, using the original training documents as is; SWR, Stop Words Removal; WS, Word Splitting.

Training Data	RAW	SWR	WS	SWR + WS
v1.0 - original	27.5, 18.1	37.0, 21.3	35.8, 20.0	46.3, 21.6
v1.0 - code	33.7, 20.5	34.9, 20.6	38.2, 20.3	41.7, 21.1
v1.0 - text	34.5, 20.3	**57.3, 21.3**	33.8, 20.3	56.3, 21.1
v2.0 - original	49.5, 12.3	60.4, 12.0	55.8, 12.3	**62.7, 11.7**
v2.0 - code	56.2, 11.8	57.1, 12.0	61.9, 11.7	61.4, 11.6
v2.0 - text	60.0, 12.6	62.2, 11.7	62.1, 11.6	62.2, 11.6
v3.0 - original	92.1, 3.0	**93.6, 2.6**	91.3, 3.0	91.8, 3.0
v3.0 - code	91.5, 3.0	92.1, 2.9	89.7, 3.2	89.6, 3.2
v3.0 - text	92.1, 2.9	93.0, 2.8	92.0. 3.0	92.7, 2.8

text categorization if they are used as is, because each of them will be counted as a different word. To address this problem, we have implemented a simple words splitting algorithm to preprocess the API discussions before they are sent to the learning algorithm. The words splitting algorithm, for example, would split "`set_value`" into two words, "`set`" and "`value`", "`ImageIcon`" into "`image`" and "`icon`", "`actionPerformed`" into "`action`" and "`performed`", and "`button3`" into "`button`" and "`3`". Apparently, many of these words are meaningful for our text categorization task. The words splitting algorithm increases the frequencies of these words and, thus, may have some impact on the classification accuracy, which we have tested empirically.

3) Using Code or Text Alone: To experiment with the effects that code may have on classification accuracy, as shown in Table II, for each version of the training data, we also create a version that contains the quoted source code only and a version that contains the text only.

D. Experimental Results and Observations

Our first research question **RQ1** is to find out if the Naive Bayes algorithm can help us achieve an adequate classification accuracy. To seek a satisfactory answer for **RQ1**, we have created three versions of training data. From each version, we have extracted two more sets of training data, which consist of only code and only text from the original discussion documents, respectively. We tested each set of training data with four different feature selection methods (using the original documents as is (**RAW**), with stop words removed (**SWR**), with words splitting (**WS**), and with the combination of stop words removal and words splitting (**SWR+WS**). See Section IV-C for details on feature selection.

Table II shows a summary of all the experiments that we have run for **RQ1**. All experiments were run using the MALLET commands and evaluation metrics described in Section IV-B. For each experiment, we show the classification accuracy mean and the associated standard deviation, which measures the amount of variance in the resulting accuracy.

Based on the experimental results shown in Table II, we can make the following observations:

1) “Stop Words Removal” is always helpful for getting better training results. Its effect of improvement is more noticeable when applied to smaller training samples (Versions 1 and 2) and almost negligible for the large training sample (Version 3).
2) “Words Splitting” is helpful for getting better training results when applied to the smaller training samples (Versions 1 and 2). It does not help for the large training sample (Version 3).
3) When the sample size is small (Versions 1 and 2), both “Code Only” data and “Text Only” data help improve classification accuracy. However, they are not helpful for the large training sample, although their accuracies are very close to that of the original documents (Version 3).
4) Overall, an adequate training sample size is perhaps the single most effective factor for improving the test accuracy mean TAM.

E. A Theoretical Interpretation for Classification Accuracy

As shown in Section III, in a probabilistic learning framework, the goal is for the learning algorithm to estimate the actual conditional probability $P(C_i \mid D)$ based on information obtained from the labeled training instances. In [6], Friedman analyzes in detail the ways by which *estimation bias* (the difference between the actual and the estimated conditional probabilities) and *estimation variance* (the variance of the estimated conditional probability) together influence classification error and accuracy. In particular, Friedman shows why under certain conditions, the classification algorithm may still work very well, despite the presence of significant estimation bias caused by the crude estimation that a simple algorithm such as the Naive Bayes produces [6].

Friedman also concludes that estimation variance (the variance of the estimated conditional probability) is critical for controlling classification error/accuracy: The lower the variance, the higher the classification accuracy [6]. This theory of estimation variance is particularly important and relevant for this paper as it appears to be consistent with and can be used to explain our experimental results and observations in Section IV-D.

One important way to reduce probability estimation variance is to increase the training sample size [6]. This method has been used previously, but only empirically and implicitly without sound rationale, e.g., [16]. As can be seen from Table II, when the training samples (Versions 1 and 2) are not big enough, the classification accuracy tends to be low with higher variances (compare the standard deviations). By contrast, the largest training sample, Version 3, produces the highest classification accuracy with the smallest stddev.

“Stop Words Removal” helps reduce the size of the classifier’s vocabulary. As a result, its application would help reduce the estimation variance and, thus, increase classification accuracy (See Equation 4). However, As can be seen from

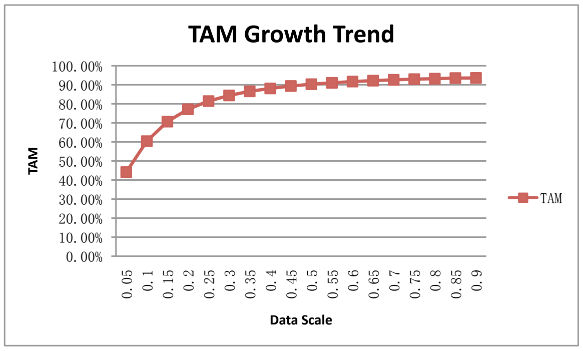


Fig. 2: Average Test Accuracy (TAM) as a function of training set size

Table II, its effect for classification accuracy is robust across all versions, but more noticeable for smaller samples.

The application of “Words Splitting” would increase the frequencies of certain words. For smaller samples, this seems to have helped reduce the estimation variance and, thus, increase classification accuracy. However, understandably, for the largest training sample, its application shows no positive effect for classification accuracy.

Like others [16], we have not applied stemming to our training samples. Based on Friedman’s theory [6], we predict that the effect of stemming will be similar to “Words Splitting” since in general, stemming also increases word frequencies.

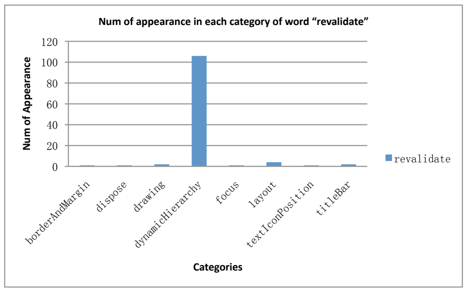
F. Training Set Size versus Classification Accuracy

As shown in Table II, we were able to achieve an average test accuracy as high as 93.6% with the Version 3 training data. However, our experience with manually collecting 833 documents for Version 3 as well as that of others is that it is costly to create large, high-quality training data for machine learning [18]. Therefore, it becomes only natural to ask the question whether the 833 documents in Version 3 are all necessary and what is the minimal sample size that can still yield a comparable classification accuracy.

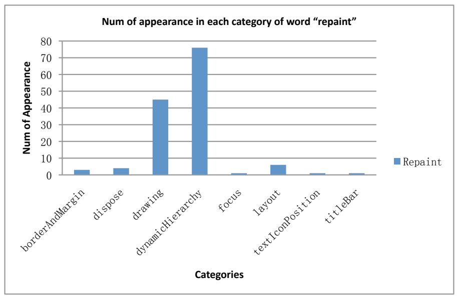
To find out, we ran the Naive Bayes algorithm 18 times using the second command in Section IV-B, varying its training portion from 5% to 90% with an increase of 5% each time. Figure 2 depicts the relation between the training set size and Test Accuracy Mean (TAM). It shows that the TAM is increasing when the training data grow from 5% to 50%. However, after the training data portion reached 50%, that is, when 417 documents are used for training, the classifier tends to perform reliably with a TAM over 90%. It appears that the other 40%, or 333 documents, increase the TAM by only less than 4%. For our task, if an above-90% is considered acceptable and the cost of collecting training data is a concern, we would have been able to achieve an above-90% test accuracy mean with 480, or 60 documents per category.

G. Feature Word Frequencies versus Classification Accuracy

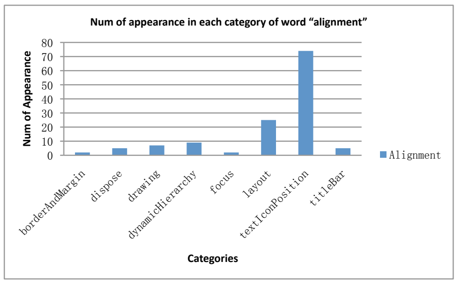
The highest average test accuracy in Table II is 93.6%, implying that there is an average error rate of 6.4%, or



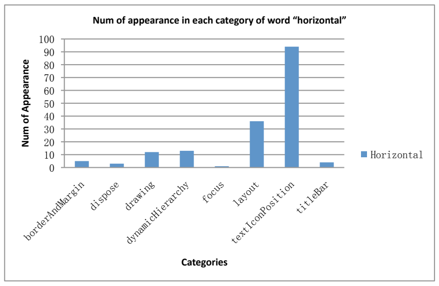
(a) "revalidate"



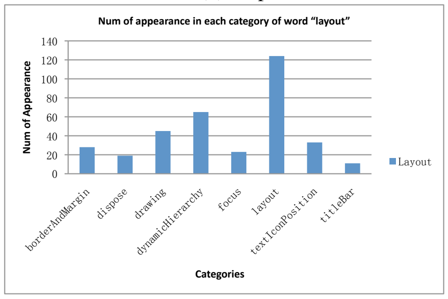
(b) "repaint"



(c) "alignment"



(d) "horizontal"



(e) "layout"

Fig. 3: Frequency distributions of five selected feature words over the eight categories in Version 3

that on average, five documents (6.4% of the 833*10%, or 83 test instances) are incorrectly classified. To investigate what may have caused the errors as well as to confirm our theoretical understanding on how Naive Bayes works (cf. Section III), we've selected five words that we suspect should have contributed higher discriminative power for separating the categories. We compare their occurrence frequencies in each category. As a result, we have found two related causes for prediction errors. One is that while some of these words are indeed highly discriminative, the more general case is that a category tends to be defined by multiple words, and the same word may appear in more than one category, leading to classification errors. The other finding is that a discussion may inherently involve multiple categories/topics. We were also able to demonstrate a solution to reduce the errors.

The five selected words are "`revalidate`," "`repaint`," "`alignment`," "`horizontal`," and "`layout`." Figure 3 shows their frequency distributions over the eight categories.

- Discussions in Category dynamicHierarchy are about how to dynamically update a GUI, for which the API method `revalidate()` must be called in order to calculate the layout of a changed GUI hierarchy. Figure 3a shows that the word "`revalidate`" is a distinct feature that helps clearly separate Category dynamicHierarchy from others.
- The word "`repaint`" is related to dynamicHierarchy as well. The Swing API method `repaint()` must be called to make the changed GUI hierarchy visible on screen. However, as can be seen from Figure 3b, this word is commonly used in not only the "dynamicHierarchy" category, but also the "drawing" category.
- Discussions in Category textIconPosition are about how to manipulate the relative positions between the text and the icon in a button or a label, which can be aligned either horizontally or vertically [9]. As a result, both words "`alignment`" and "`horizontal`" appear in this category. Figures 3c and 3d depict the occurrence distributions of these two words. However, people may also discuss how to align GUI components in other categories, especially in the "layout" category. Similarly, the word "horizontal" is often used in the layout category to describe the direction of laying out a set of components.
- Discussions in the layout category are about how to compose and position multiple widgets in order to make a graphical user interface [19]. Figure 3e depicts the frequencies by which the word "`layout`" appears in the eight categories. Interestingly, different from the other four, in addition to Category layout, the word `layout` also appears in all of the other seven categories.

Why is the word `layout` contained in all eight categories in Figure 3e? This is likely due to the fact that layout is a basic feature required by almost all Swing programs. Even for a discussion that is not centered on layout, as long as it quotes some source code, the word `layout` is likely to be included. Furthermore, the word may also be used when the posters explain how the program under discussion works.

We have also observed that documents from other categories are most often mistakenly classified into the layout category. We hypothesize that this is due to the combined effects of the common occurrences of the word `layout` and the quoted source code. In particular, to test the hypothesis that the quoted code may have caused documents from the other categories to be categorized as layout, we created a new dataset for Version 3, where we used the "Text Only" data for the layout category in order to remove the confusion due to the quoted code. We

TABLE III: Test Accuracy Mean (TAM) and the associated stddev for the N-label Naive Bayes Classifier

#labels	TAM, stddev
N = 2	97.8, 1.6
N = 3	99.3, 0.9
N = 4	99.7, 0.5

trained a new Naive Bayes classifier with stop words removed. When tested, the new classifier yielded an improved average test accuracy of 94.1%, which is 0.48% higher than the best average test accuracy 93.6% in Table II.

H. Multi-label Text Categorization

So far, we have focused on the case in which exactly one category is assigned to each document, which is often called the *single-label* (aka *non-overlapping categories*) case [20]. In Section IV-G, we have also seen the case in which multiple categories may be assigned to the same document, which is dubbed the *multi-label* (aka *overlapping categories*) case [20].

When treated as a single-label case, a document that inherently involves more than one category (topic) would naturally cause a prediction error. In Version 3 of our training data, we have 833 documents, of which 10% (83 documents) are used for testing. Since the highest Test Accuracy Mean (TAM) is 93.6% (Table II), in each trial, there are on average about 5 documents (6.4% * 83) wrongly predicted. To investigate the impact that the multi-label problem may have on classification accuracy, we have modified the MALLET's Naive Bayes algorithm to output the top-N categories that yield the highest conditional probabilities $\bar{P}(C_i \mid D)$, rather than only the top category with the highest probability. Under this modification, a classification decision is considered correct as long as the N categories that it predicts include the manually assigned label for the test document.

Table III shows the classification accuracy for N equals 2, 3, and 4, respectively. It appears that small values for N are enough to drive the prediction accuracy to a nearly perfect score. This is promising as it shows that the multi-label approach may be acceptable for practical use in Software Engineering. This is because on average, out of every N documents, there is at least one relevant; a software developer would not mind to inspect a few additional documents so long as he or she is sure that at least one of them is relevant.

I. Two Examples of Multi-label API Discussions

In this section, we present two sample API discussions to illustrate the multi-label phenomenon. To simplify presentation, we have edited the discussions to highlight only the most relevant pieces. However, the two original discussions are also available online for further inspection [4] [5].

The first sample discussion involves both the layout category and the drawing category. It was manually labeled as layout, but the learned Naive Bayes classifier classified it as drawing.

[4]https://forums.oracle.com/forums/thread.jspa?messageID=5865516
[5]https://forums.oracle.com/forums/thread.jspa?messageID=5861973

In the first message shown next, the OP (original poster) asks for a solution and also posts code for other participants to review. The original task is to figure out the reason why his canvas and buttons are displayed on the left hand side. From the code we can see the OP was using "borderLayout" and trying to add panels to the mainContainer. But since the OP does not set the border layout correctly, the program positions the "canvas" in the wrong place. Therefore, the OP's problem is about how to correctly set up a layout manager. So this is indeed a "layout" topic.

```
Message Title: Layout Problems

Can someone help me understand why my
canvas here shows up on the left hand
side along with my buttons.
```

However, in addition to layout-related words, the posted code also includes a set of keywords that are commonly used when describing drawing issues, such as `repaint`, `Graphics`, `setX`, `draw`, and `repaint`. So the discussion is also about drawing, although the central topic is layout instead of drawing.

```
public class Painter extends JFrame ...{
...
// Adding panels
mainContainer.setLayout BorderLayout;
mainContainer.add Buttons, BorderLayout;
mainContainer.add BorderLayout CENTER;
...
void clear(){  repaint(); }

public actionPerformed (ActionEvent e){
... repaint(); ...
}
public mousePressed (MouseEvent e){
  myShape.setX1(e.getX());
  myShape.setY1(e.getY());
}
public paint(Graphics g){
  g.setColor(line);
  myShape.draw(g);
 }
}
```

The last post in the discussion, which is shown next, would contribute more feature words to the layout topic, e.g., `center`, `layout`, `north`, and `components`.

```
Re: Layout Problems
...
well, since you dont' seem to put
anything in the ``canvas'', I dont'
see how you know that it's doing
what you say...

I can tell you that since it's in the
```

```
center of the border layout, it fills
whatever space isn’t used by the
north, south, east and west components
(if any).
```

Our second sample API discussion involves both the topics of borderAndMargin and rendererEditor. The discussion was manually labelled as rendererEditor, but the classifier assigns it to the borderAndMargin category. The OP’s task is to find out how to enhance a selected cell in a JTable. which can be implemented by customizing the renderer of the JTable.

```
Message title:  cell renderer with border
...
Dear friends,
every time I apply a cell render over a
JTable, the selected row is not enhanced
... why ?
```

The next piece of code posted would contribute to the borderAndMargin topic due to the use of feature words such as `setBorder` and `BorderFactory`:

```
Re: cell renderer with border
...
into the tutorial I found the following:
if (isBordered) {
    if (isSelected) {
    if (selectedBorder == null) {
    selectedBorder = BorderFactory.
         createMatteBorder(2,5,2,5,
         table.getSelectionBackground());
    }
    setBorder(selectedBorder);
    } else {
      if (unselectedBorder == null) {
      unselectedBorder = BorderFactory.
         createMatteBorder(2,5,2,5,
         table.getBackground());
    }
    setBorder(unselectedBorder);
    }
}
Thats working fine, but it only enhance
the row, not the selected cell. ??
How to obtain the ``normal'' selection
effect - i.e., that one where I not using
renderers?
```

But the next piece of code posted clearly contributes to the rendererEditor category:

```
Re: cell renderer with border
...
How to obtain the ``normal'' selection
effect - i.e., that one where I not using
renderers?

Extend the DefaultTableCellRenderer:
class MyRenderer extends
DefaultTableCellRenderer
{
  public Component
  getTableCellRendererComponent(...)
  {
   super.getTableCellRendererComponent...
    // add your code here
    return this;
  }
}
```

The last two messages would also contribute to the borderAndMargin topic due to the occurrences of feature words such as `alignment`, `setAlignmentX`, and `setHorizontalAlignment`:

```
Re: cell renderer with border
...
thank you very much, everything is
``almost ok'' now :))
the problem now is the cell alignment...
Im trying:
setAlignmentX(JLabel.RIGHT_ALIGNMENT);
but the cell doesnt change it alignment
in runtime... why ?

Re: cell renderer with border
...
I believe that should be
setHorizontalAlignment(JLabel.RIGHT);
```

J. Experimentation with Support Vector Machines

Although we have primarily focused on an in-depth investigation of the Naive Bayes, it is also relevant to compare it with other machine learning algorithms, such as Decision Trees and SVM (Support Vector Machines). SVM, in particular, has been shown to be very effective for text classification [10]. Since our Version 3 data is readily available, we have also tried out SVM for a quick comparison. In general, SVM is shown to outperform Naive Bayes by one to four percent in terms of the average test accuracy (compare Tables II and IV).

The Naive Bayes is a multi-category classifier (eight in our case), whereas SVM classifiers are binary. As a result, we have trained one SVM classifier for each of the eight categories. We took 10 percent of instances from each category to make a test set, and used the other 90 percent to make the training datasets. We applied the leave-one-out cross validation approach.

When making a training dataset for each SVM classifier, the negative instances far outnumber the positive instances, so such a dataset is considered *imbalanced* [2]. When trained with such imbalanced datasets, the performance of SVM has been shown to drop significantly [2]. To create a balanced dataset, we have preprocessed the data by *oversampling* the positive class (duplicating the positive instances such that their number matches that of negative instances) and *undersampling* the negative classes (proportionally reducing the size

TABLE IV: Performance of eight binary SVM classifiers trained with three datasets derived from Version 3: Imbalanced, Undersampling, and Oversampling. Each cell contains Accuracy/Precision/Recall on test set for the respective binary classifier.

Classifiers	**Imbalanced**	**Under-sampling**	**Over-sampling**
borderAndMargin	93.41% / 75.00% / 60.00%	87.91% / 47.06% / 80.00%	98.99% / 100.0% / 90.91%
dispose	95.60% / 100.0% / 60.00%	86.81% / 45.45% / 100.0%	96.97% / 100.0% / 72.73%
drawing	98.90% / 100.0% / 90.91%	97.80% / 90.91% / 90.91%	99.07% / 100.0% / 90.91%
focus	96.70% / 100.0% / 72.73%	96.70% / 90.00% / 81.82%	96.97% / 90.00% / 81.82%
layout	93.41% / 85.71% / 54.55%	92.31% / 66.67% / 72.73%	96.00% / 76.92% / 90.91%
titleBar	93.41% / 72.73% / 72.73%	93.41% / 72.73% / 72.73%	94.05% / 75.00% / 81.82%
textIconPosition	96.70% / 100.0% / 72.73%	100.0% / 100.0% / 100.0%	98.98% / 100.0% / 90.91%
dynamicHierarchy	95.60% / 100.0% / 63.64%	81.32% / 39.29% / 100.0%	98.98% / 100.0% / 90.91%
average	**95.47**% / 91.68% / 68.41%	**92.03**% / 69.01% / 87.27%	**97.50**% / 92.74% / 86.37%

of each negative class such that the sum of all negative classes matches that of the positive class). Table IV depicts the accuracy/precision/recall of the SVM binary classifiers trained with the three datasets (imbalanced, undersampling, and oversampling). As shown in Table IV, oversampling is the best strategy that yields the highest average test accuracy, whereas undersampling the worst.

V. Conclusion and future work

Software libraries and APIs are important productivity tools, but they are difficult to use due to their extensive content and rich details. As a result, developers have widely used online software forums to exchange information and seek help to solve problems in using APIs. Tens of thousands of discussions have been archived for popular frameworks such as AWT/Swing, which can be referenced by future developers to improve productivity. However, manually finding relevant discussions from the vast amount of discussion threads is a painstaking task for the user. Therefore, it can be useful to automatically classify these discussions into meaningful semantic categories so as to facilitate searching and browsing.

To address this problem, we have conducted an exploratory study where we utilized the Nave Bayes algorithm in the MALLET machine learning toolkit to categorize API discussions into API-specific topics. In our case study, we analyzed over 1,000 discussion threads from the Java Swing Forum. We labeled these API discussions with predefined semantic categories based on their textual content. We've built three groups of training data consisting of 46, 158, and 833 documents, respectively. With our largest training data set (833 documents over eight categories), we have successfully trained Naive Bayes classifiers that achieve the highest average test accuracy of 94.1%. In addition, we investigated the impacts of various feature selection methods on classification accuracy, including stop words removal, words splitting, "Code Only," and "Text Only." We found that the size of training set is a key factor for improving classification accuracy. We have also identified that multi-label documents are a main cause for classification errors and proposed one solution. Lastly, we investigated why Naive Bayes works so well by inspecting the frequency distributions of selected words from our training data, and by seeking theoretical interpretation from the machine learning literature. Future work should apply the same process to other APIs to demonstrate generality.

References

[1] stackoverflow. [Online]. Available: http://stackoverflow.com/

[2] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *ECML*, 2004, pp. 39–50.

[3] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *CASCON*, 2008, pp. 23:304–23:318.

[4] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *ICSE*, 2006, pp. 361–370.

[5] A. Bacchelli, T. D. Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *ICSE*, 2012, pp. 375–385.

[6] J. H. Friedman, "On bias, variance, 0/1-loss, and the curse-of-dimensionality," *Data Min. Knowl. Discov.*, vol. 1, no. 1, pp. 55–77, Jan. 1997.

[7] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *ASE*, 2011, pp. 323–332.

[8] A. Hindle, D. M. German, R. C. Holt, and M. W. Godfrey, "Automatic classification of large changes into maintenance categories," in *ICPC*, 2009, pp. 30–39.

[9] D. Hou and L. Li, "Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions," in *ICPC*, 2011, pp. 91–100.

[10] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *ECML*, 1998, pp. 137–142.

[11] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, Mar. 2008.

[12] P. Langley, W. Iba, and, and K. Thompson, "An analysis of bayesian classifiers," in *AAAI*, 1992, pp. 223–228.

[13] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *ECML*, 1998, pp. 4–15.

[14] D. D. Lewis and M. Ringuette, "A comparison of two learning algorithms for text categorization," in *Third Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 81–93.

[15] A. McCallum and A. Kachites. (2002) MALLET: A machine learning for language toolkit. [Online]. Available: http://mallet.cs.umass.edu

[16] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on "Learning for Text Categorization"*, 1998.

[17] A. Murgia, G. Concas, M. Marchesi, and R. Tonelli, "A machine learning approach for text categorization of fixing-issue commits on CVS," in *ESEM*, 2010, pp. 6:1–6:10.

[18] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Machine Learning*, vol. 39, pp. 103–134, 2000.

[19] C. R. Rupakheti and D. Hou, "Evaluating forum discussions to inform the design of an API critic," in *ICPC*, 2012, pp. 53–62.

[20] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[21] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *ICSE*, 2010, pp. 45–54.

[22] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *SIGIR*, 1999, pp. 42–49.