Categorizing API Forum Discussions

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

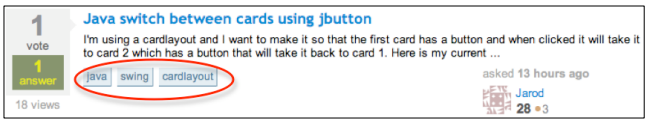
Lingfeng Mo
Department of Computer Science
Clarkson University
Potsdam, New York 13699
Email: mol@clarkson.edu

***Abstract*—*Text categorization*, automatically labeling natural language texts with pre-defined categories based on their content, is an essential task for efficiently managing the abundant online data. Within Software Engineering, an example of such data is the ever-growing, large volume of forum discussions about using 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. Our goal is two-fold: (1) Can a relatively straightforward algorithm such as Naive Bayes be made to work sufficiently well for this task? (2) If yes, how can we control its performance? We achieve the best test accuracy mean of 93.6% with our largest training data set, which consists of 833 forum discussions distributed over eight categories/topics, for the AWT/Swing API. We also investigate several factors that impact on classification accuracy. Among others, we find that the size of the training set and multi-label documents (that is, the phenomenon that some discussions involve more than one category) are the key ones.**

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

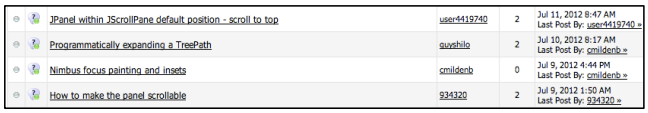
I. Introduction

Nowadays, online forums are commonly used by software developers to exchange information with each other, asking questions and seeking help. Over time, these forums have become repositories of valuable programming tips and knowledge that many developers learn to revisit again and again. To facilitate the browsing and searching of software forums, however, we need to find more semantic information to index the forum data beyond just keywords, for example, the tags used by Stack Overflow [1], as what is shown below.



However, Stack Overflow relies on users to assign tags manually. *Text categorization*, automatically labeling natural language texts with pre-defined categories based on their content, can be applied to automate this task [17].

In this paper, we investigate *text categorization* as a fundamental approach to add the needed semantic information by tagging the 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.



Our goal has been two-fold: First, we want to investigate the feasibility of using existing learning algorithms to categorize online forum discussions. Second, we want to understand exactly how a learning algorithm works for our task.

We chose to start with the Naive Bayes algorithm [11] to categorize the API discussions in the Swing Forum [1]. Historically the Naive Bayes algorithm has been found performing "remarkably well" for many tasks [10], [12]. Although more recent studies have compared it with other algorithms [19], [8], 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.

We chose the Swing Forum because it is a very active forum; as of July 14, 2012, it contains 47,132 discussion threads with 212,199 messages. Another advantage of choosing Swing Forum is that we can leverage the considerable prior research that we have conducted with the forum, e.g., [7], [16], so we can have some confidence with the quality of our training data. We treat a whole discussion thread, which typically has multiple messages/posts, as a training document.

The contributions of this work are summarized as follows:

- Publicly available training data sets: To experiment with the Naive Bayes machine learning algorithm, 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 Swing. We have made our data publicly available [2].
- Using our data, we show that the Naive Bayes classifier can achieve on average a classification accuracy as high as 93.6%.
- 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 July 14, 2012)

[2] **http://www.clarkson.edu/~dhou/projects/swingForum2012.tar.gz**

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

II. Related Work

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

A. Software Forums

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 ad hoc analysis techniques for each category [4].

Gottipati, Lo, and Jiang present a tag inference algorithm that automatically assigns seven different tags to posts: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk [5]. Unlike ours, their unit of classification is a message/post, rather than a whole discussion thread. Furthermore, their tags are about the conversational role that a message/post plays in a discussion, rather than the API-specific semantics of a whole discussion.

In this study, we have leveraged two sets of Swing Forum discussions that we collected and analyzed as part of two of our own prior research projects [7], [16]. The first set consists of 172 discussions, which were analyzed to understand the kinds of API obstacles programmers may have in practice [7]. 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 [16].

B. Bug Repositories

Bug reports in large projects must be triaged to determine if a report requires attention and if it does, which developers can be assigned the responsibility of resolving the report. To ease the task of assigning reports to developers, Anvik, Hiew, and Murphy apply a machine learning algorithm to the open bug repository to learn the kinds of reports each developer resolves [3]. When a new report arrives, the classifier suggests a small number of developers suitable to resolve the report.

A related problem in bug repositories is triaging duplicate bug reports. Sun et al. present a machine learning approach to classify whether a new bug report is a duplicate [18].

Antoniol et al. study how machine learning algorithms can be used to separate true bug reports from other change requests [2]. They find that alternating decision trees, naive Bayes classifiers, and logistic regression can be used to accurately distinguish bugs from other kinds of issues.

C. Source Control Systems Change Classification

Hindle et al. present a study to automatically classify large changes into categories of maintenance tasks - corrective, adaptive, perfective, feature addition, and non-functional improvement - using machine learning techniques [6]. They find that for most large commits, the commit messages plus the commit author identity contain enough information to accurately 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. use a machine learning classifier to determine whether a new software change is more similar to prior buggy changes, or clean changes [9].

III. The Naive Bayes Algorithm

We used the MALLET machine learning toolkit for language processing [13]. In what follows, we briefly review the Naive Bayes algorithm. More details about it can be found elsewhere [11], [14], [15], [10].

The Naive Bayes algorithm learns a classifier from a training sample 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 $P(C_i \mid D)$ for D. More 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 III, 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, the Naive Bayes algorithm assumes that the n terms be independent of their 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 [14], [11] but with some combinatorial coefficients removed for simplification. Moreover, for natural languages, the Naive Bayes assumption of course does not hold in general. But in practice, despite this unsound assumption, this algorithm has been shown to work sufficiently well for many applications [10].

TABLE I: The Version 3 training data

Data Version	Categories/Labels	#Documents	Total
V3.0	borderAndMargin	82	833
	dispose	103	
	drawing	108	
	focus	104	
	layout	125	
	titleBar	106	
	textIconPosition	96	
	dynamicHierarchy	109	

IV. Experimental Process and Results

A. Data Collection and Evaluation Metrics

The most time-consuming task in text categorization is preparing the training data [15]. In our case, unlike other studies [2], [6], the categories are domain-specific and do not exist beforehand. Therefore, we need to come up with both the categories and collect the training documents for each category. In addition, we were not sure what would be the minimum number of documents that each category must have in order to achieve reliable, good-enough learning. Consequently, we have evolved three versions of training data [3]. Table II summarizes the classification accuracy for all of our experiments.

Version 1 was created quickly to test the 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 test accuracy mean 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. Although the accuracy for Version 2 is not adequate either, the improvements are noticeable.

Encouraged by the improvement in Version 2, in Version 3 we aggressively increased the sample size by about ten times to about 100 documents per category. Table I depicts the Version 3 training data. Note that the categories in Table I are specific to the Swing API. For example, the border category discusses problems related to setting borders for Swing widgets, and the dispose category about ways to properly close a window.

We used the MALLET machine learning toolkit for language processing [13]. Specifically, MALLET was configured to randomly split the training data into 90% training (-and-validating) instances for learning the classifier, and 10% for testing. This training-and-testing process was repeated 10,000 times, or trials. The test accuracy for each trial is defined as the ratio between the number of correctly labeled instances and the total number of test instances. An overall measure of classification accuracy, TAM (Test Accuracy Mean), is calculated, as the mean of the 10,000 test accuracies. Table II summarizes the classification accuracy for all of our experiments.

[3]All of our data are single-labeled.

TABLE II: Test Accuracy Mean (TAM) and 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, Words 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

B. 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," also known as stop words, do not contribute to the discrimination power needed for text categorization and, thus, can be removed. The vocabulary size for Version 3 is reduced from 9,617 to 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. 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 increases the frequencies of domain 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.

C. Experimental Results and Observations

Table II shows a summary of all the experiments that we have run. For each experiment, we show the classification accuracy mean and the associated standard deviation, which measures the amount of variance in the resulting accuracy. We can make the following observations based on Table II:

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).

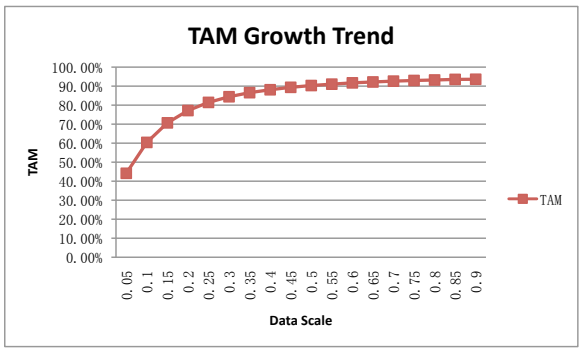


Fig. 1: Test Accuracy Mean as function of training set size

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

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.

D. 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, it can be costly to create large, high-quality training data for machine learning [15]. 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 similar classification accuracy. To find out, we ran the Naive Bayes algorithm 18 times, setting its training portion from 5% to 90% with an increase of 5% each time. Figure 1 depicts the relation between the training set size and Test Accuracy Mean (TAM). It obviously 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, we would have been able to achieve it with a total of 480, or 60 documents per category.

E. Multi-label Text Categorization

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, on average about 5 documents (6.4% * 83) are 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. With 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. 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. If this is true, the multi-label approach may be promising to be accepted for practical use in Software Engineering.

REFERENCES

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

[2] 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.

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

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

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

[6] 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.

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

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

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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