Rule-based Problem Classification in IT Service Management

Yixin Diao, Hani Jamjoom, David Loewenstern
IBM Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
Email: {diao|jamjoom|davidloe}@us.ibm.com

Abstract

Problem management is a critical and expensive element for delivering IT service management and touches various levels of managed IT infrastructure. While problem management has been mostly reactive, recent work is studying how to leverage large problem ticket information from similar IT infrastructures to probatively predict the onset of problems. Because of the sheer size and complexity of problem tickets, supervised learning algorithms have been the method of choice for problem ticket classification, relying on labeled (or pre-classified) tickets from one managed infrastructure to automatically create signatures for similar infrastructures. However, where there are insufficient pre-classified data, leveraging human expertise to develop classification rules can be more efficient. In this paper, we describe a rule-based crowdsourcing approach, where experts can author classification rules and a social networking-based platform (called xPad) is used to socialize and execute these rules by large practitioner communities. Using real data sets from several large IT delivery centers, we demonstrate that this approach balances between two key criteria: accuracy and cost effectiveness.

1 Introduction

Improving service quality in IT infrastructure management continues to be a critical driver for business growth. While there are many aspects to achieving higher quality, proactive problem management is playing a vital role as it strives to minimize the occurrence of incidents and problems, and supports the overall Incident, Problem, and Change (IP&C) management processes.

The IT Information Library (ITIL) defines problem management as the process of minimizing the adverse impact of incidents and problems, and preventing their recurrence [1]. In ITIL terminology, an incident is any event that is not part of the standard operation and results in an service interruption, and a problem is the underlying cause of one or more incidents. There are two aspects in the problem management process. Reactive problem management is concerned with solving problems in response to the incidents. Proactive problem management seeks to identify the cause and trend of the incidents, and instigate remedial actions before incidents occur in the first place. Considering outsourced environments where the service provider is managing a large number of IT infrastructures for different clients, proactive problem management can have a multiplicative benefit as learning from one IT environment can be easily replicated to similar, but not necessarily identical environments [2].

Since incidents and problems can occur at different levels of the software and hardware stack, one major activity to facilitate proactive problem management is to create leading indicators (a.k.a., signatures or failure codes), which are used to classify incidents into groups for root cause analysis and failure trend monitoring. Some example incident categories include: application not available, disk-usage threshold exceeded, system down, printer not printing, and password reset. The incident categories is usually determined through analyzing the incident records (a.k.a., problem tickets), which store incident details such as client name, platform, failure descriptions, severity code, resolution approaches, and various time stamps.

The problem ticket classification problem is a class of document classification problems in information science. The document is composed of natural language text some of which is structured and machine generated (*e.g.*, from event monitoring systems) and some of which is unstructured and user generated (*e.g.*, from communication with the clients). However, individual problem ticket instances are not identified as machine or user generated.

Document classification techniques typically start from textual analysis (or featurization), in which the document content is divided into features (words or phrases) and methods such as stemming, stop words, and compound term processing are typically used to generate the feature set [3].

978-0-7695-3840-2/09 $26.00 © 2009 IEEE
DOI 10.1109/CLOUD.2009.80

Afterwards, the document is classified either by supervised document classification where correct classification information is available as part of the training data, or unsupervised document classification where the classification must be done entirely without reference to external information. The former is applicable in our case, as the failure codes are predefined in the problem management process. Various machine learning and pattern recognition methods have been studied in supervised document classification, such as naïve Bayes [4], TD-IDF [5], support vector machines [6], kNN [7, 8], decision trees [9, 10], and neural networks [11].

Either or both of the above two steps (featurization and classification) can be conducted automatically or manually, and there have been a small number of studies comparing expert-generated rules to supervised learning. Depending upon the domain, choice of representation, definition of success, quality of data, time constraints, costs imposed in knowledge acquisition, and costs imposed in cleaning and labeling data, "automated" supervised learning can outperform "hand-crafted" expert systems [12], or vice-versa [13]. It is still true that much supervised learning is used not to outperform expert-generated rules but to substitute for them, either to open the "knowledge acquisition bottleneck" [14] or because the goal is to learn something beyond current expertise (or there is no expertise) as in data mining [15].

In this paper we propose a rule-based crowdsourcing approach with the objective of providing an effective and scalable method for supporting proactive problem determination in a global service delivery environment. The paper makes the following three contributions: (1) a simplified rule-based classification method that lowers the costs of knowledge acquisition as well as the costs of data cleaning and labeling, (2) an xPad crowdsourcing platform for increasing the reachability and consumability of classification by large practitioner communities, and (3) a detailed experimental comparison between hand-crafted rule set and supervised learning with focus on classification accuracy and cost effectiveness.

The remainder of this paper is organized as follows. Section 2 overviews the problem ticket classification problem and its challenges in IT service management. In Section 3 we present the proposed rule-based classification approach and the rule lifecycle management framework. Section 4 discusses the architecture and prototype of the xPad crowdsourcing platform. Section 5 shows the experimental evaluation results using real data from several large IT delivery centers. Our conclusions are contained in Section 6.

2 Proactive Problem Management

Figure 1 shows the process and operations of problem management in a service delivery center. The service clients interact with the service provider through ticketing systems, which coordinate ticket creation and access. The ticket records are stored in the database and contain incident details such as client name, platform, failure descriptions, and severity code. System administrators fetch the tickets from the ticketing systems, identify and solve the problems in response to the incidents, and close the tickets with documented resolution approaches. Periodically, quality analysts acquire a batch of tickets from the ticketing systems, classify the tickets by assigning the failure code to each ticket, analyzing the volume and trend of each failure class, identifying the root causes, and taking remedial actions to prevent or reduce incident recurrence.

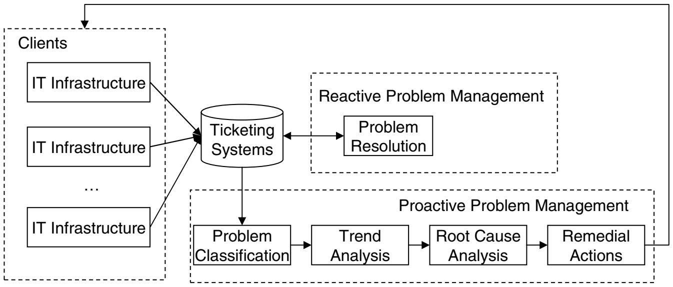


Figure 1. Illustrative process and operations of problem management.

Due to the high ticket volume, manually classifying each ticket is time-consuming and sometimes impractical. The ticket classification problem is essentially learning (or manually crafting) a function from training data, where the input is the incident record and the output is the class label of the incident. After having seen a number of training examples (*i.e.*, pairs of incident and failure code), the learned function will automatically classify the tickets and thus improve the productivity of the quality analysts. It may also improve the quality of problem classification from the perspective of consistency.

The problem management process requires the accurate and comprehensive recording of incidents in order to identify the cause of the incidents and the trends, and instigate remedial actions. In practice, however, the problem tickets are typically not well documented regarding both the incident descriptions and the resolution techniques. This is especially true in a global delivery environment where problem tickets are generated from a large number of IT infrastructures for multiple clients. Moreover, there are insufficient pre-classified tickets, which adversely impact the quantity and quality of the training data. Since labeling data is not cheap, sometimes the quality analysts apply the sampling approach by labeling only a small set of tickets, which reduces the available training examples. On the other hand, poor data quality may suffer from the fact that not all the labeled tickets are classified correctly, since the expertise of the quality analysts varies.

There are other challenges arising from the scale of global deployment. First, different delivery centers generally have a different mix of tickets and it is difficult to drive data consistency. As different clients can use different event monitoring systems, the machine-generated tickets are dissimilar across delivery centers. This variation in vocabulary can also come from humans. Although even in a single deployment, the same incident and resolution may always be described differently and the same type of tickets may be classified differently due to the user's own interpretation of the failure code, this difference is magnified in global deployment. Second, the ticket composition evolves over time, when the new clients are onboarded, new delivery centers are created, or a new failure code taxonomy is adopted to fit with a change in business requirements. All these differences and changes result in scarcity of valid training data, and require an effective means to build an adaptive classification mechanism.

In fact, some of the above challenges also indicate possible approaches to the solutions. For example, the use of machine-generated tickets means that a set of phrases associated with specific classes of tickets could be developed *a priori* to be used as features, requiring much less data than would be necessary to derive useful phrases by statistical methods. Furthermore, in this domain of problem ticket classification, there is an existing source of expertise held by the quality analysts. While it is more difficult to extract their knowledge (e.g., in the form of rules), it is also in their best interest because they can free up their time on ticket classification (especially for recurrent incidents with large volume) and focus on more intellectually challenging tasks.

While eliciting ticket classification knowledge sounds similar to creating an expert system, it is different in that expert systems typically involve three roles: domain experts to provide the knowledge, knowledge experts to create the rules, and non-experts as the expert system users. In contrast, considering the challenges and opportunities in problem ticket classification, we propose a scalable crowdsourcing approach, where all three roles are performed by the same community of the quality analysts. The rules will be designed by the quality analysts and used by the quality analysts.

This subtle change in the goal of knowledge acquisition and deployment has the potential of increasing the quality of the knowledge (since it is their own interest for getting better rules). However, it also adds the challenges on developing a simple rule-base design and management approach suitable for the mass, and leveraging the mass collaboration enabled by Web 2.0 technologies.

3 Rule-based Classification

In this section we present a design for rule-based classification. The classifier uses simple IF-THEN rules to classify the tickets, and the rules are managed and updated based on rule quality statistics. This design is crowdsourcing-oriented, making the rules simple and scalable, so that the quality analyst community can develop and use the rules easily, effectively, and cooperatively.

3.1 Classification Rules

A rule-based classifier is composed of a set of rules. Each rule is composed of a IF clause and a THEN clause. The syntax for the IF clause may be as simple as pure conjunctions of binary features (such as rules derived from decision trees and constructed by ID3[16]), or as complicated as arbitrary boolean expressions or probability statements. Similarly, the method for combining rules to reach a classification may be as simple as "the THEN clause is the classification, and the rules must be disjoint so that there is no conflict among them", or arbitrarily complex.

In this paper we take a compromise approach to permit sufficient expressive power while keeping the rules easy for the users to write and understand, noting that the users are experts in solving IT incidents and problems, but are rarely trained in knowledge engineering. In our solution, the IF clause includes patterns combined through logical ANDs or ORs to a maximum of two levels. Each pattern specifies either a word or a phrase, and a pattern matches a ticket if it shares that word or phrase with the ticket, with case sensitivity as a selectable option. The use of phrases as patterns is of particular value when creating rules to handle machine-generated text, such as error messages. The flexibility in handling of case is important for human-input incident description and solution.

The rule matches the ticket if the boolean expression of the IF clause evaluates to true where matching patterns are "true" and non-matching patterns are "false." The THEN clause then names a ticket failure code, or class. To simplify the rule base, we only consider the following two types of rules.

- Inclusive rules: The inclusive rules take the following format

  IF $(P_{11}||P_{12}||\ldots||P_{1,n_1})$ *&&* $(P_{21}||P_{22}||\ldots||P_{2,n_2})$ *&&* $(P_{31}||P_{32}||\ldots||P_{3,n_3})$ *THEN* FC_k

  where $P_{i,j}$ indicates the j-th pattern in the i-th OR group, FC_k indicates the k-th failure code, || denotes the logical OR operation, and && denotes the logical AND operation. Each inclusive rule also has a confidence score for arbitrating rule conflictions (which will be described in detail below). Note that we only use

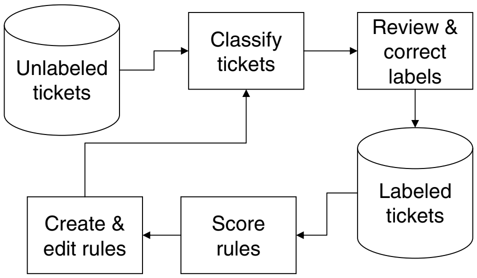


Figure 2. Interactive learning process for classification rules.

two layers of AND and OR, which in our experience has been sufficient.

- Exclusive rules: The exclusive rules take the following format

  IF ($P_{11}||P_{12}||\dots||P_{1,n}$) *THEN NOT* FC_k

  Note that the exclusive rules take a simpler format. The rules are valuablefor overruling inclusive rules by stating that certain patterns preclude certain classes. In practice, the use of exclusive rules makes up for the limited syntax of the inclusive rules.

These rule formats are chosen for ease in writing rules rather than any advantage in classification. For example, there is no featurization (pre-processing) pass: the patterns in the IF clause correspond directly to the words or phrases of tickets, but without any attempt to handle word stemming or synonyms. It was felt that the advantages of featurization in permitting the creation of a smaller number of more versatile rules while still covering the same examples did not make up for the added complexity in learning to author the rules.

Evaluation of a ticket proceeds in two passes. First, all exclusive rules are applied and the resulting failure code categories are excluded. Then, all inclusive rules for classes not already excluded are applied. The classification of the THEN clause of the inclusive rule with the highest confidence is then associated with the ticket. In the case of ties among inclusive rules with different THEN clauses, or if no rule matches, the ticket is left unclassified.

3.2 Rule Lifecycle Management

In problem ticket classification, the process of labeling data is not cheap, so we wish to generate our rule set using the smallest number of labeled tickets possible while still maintaining sufficient classification coverage and accuracy. We achieve this goal using an interactive learning process, similar to that discussed in [17]. This process, as shown in Figure 2, consists of four steps. (1) A batch of unlabeled tickets is classified by any existing rules (initially, there are none). (2) The resulting mixture of labeled and unlabeled tickets are reviewed by the user, and the labels are corrected if necessary. (3) The existing rules are scored against the labeled tickets. (4) The user reviews the rules by correcting inaccurate rules and adding new rules where coverage is poor. The above four steps iterate until the user is satisfied with the coverage and accuracy of the resulting rule set. Note that in the interest of time and productivity, the user is unnecessary to review all labels and make corrections in Step 2. Typically, examining a few labels in each interactive cycle is sufficient to identify the problematic rules or the opportunities for new rules.

We define the following two rule statistics to quantify the quality of the individual rules as well as the rule set. Deriving these statistics depends on the existence of pre-classified tickets.

- *Coverage*: The coverage metric defines the percentage of tickets that can be matched by the rules. We calculate coverage as

  $$coverage = (TP + FP)/T \tag{1}$$

  where TP (true positive) is the number of tickets in the set that the rule correctly classifies, FP (false positive) is the number of tickets in the set that the rule matches but incorrectly classifies, and T (total) is the number of tickets in the set. Note that the coverage rate is typically a metric used for the whole rule set. However, we can also use this metric to quantify the performance of an individual rule or a subset of the rules dedicated for one failure code. It this case, it generally makes more sense to define T as the number of tickets belonging to the failure code, so that the coverage is defined relative to the size of this failure code class.

- *Accuracy*: The accuracy metric defines the percentage of tickets that are correctly matched by the rules. We calculate accuracy as

  $$accuracy = TP/(TP + FP) \tag{2}$$

  Note that we define accuracy relative to the rule matches, instead of the total number of tickets in the data set. In this way, we can use accuracy as the measure of rule confidence, for example, if the number of false positive is zero, we are 100% confident of this rule.

The coverage and accuracy defined above are only for inclusive rules only, but in the presence of exclusive rules. This is because the existence of exclusive rules is to restrict

Table 1. An illustrative rule-based classifier with rule quality statistics.

		Total	TP	FN	FP	Coverage	Accuracy
FC-TOP10		500	335	165	128	93%	72%
CAP01	File system full	97	95	2	52	152%	65%
	Rule 1	97	45	52	1	47%	98%
	Rule 2	97	70	27	50	124%	58%
	Rule 3	97	37	60	10	48%	78%
CAP07	CPU / Mem	54	32	22	10	78%	76%

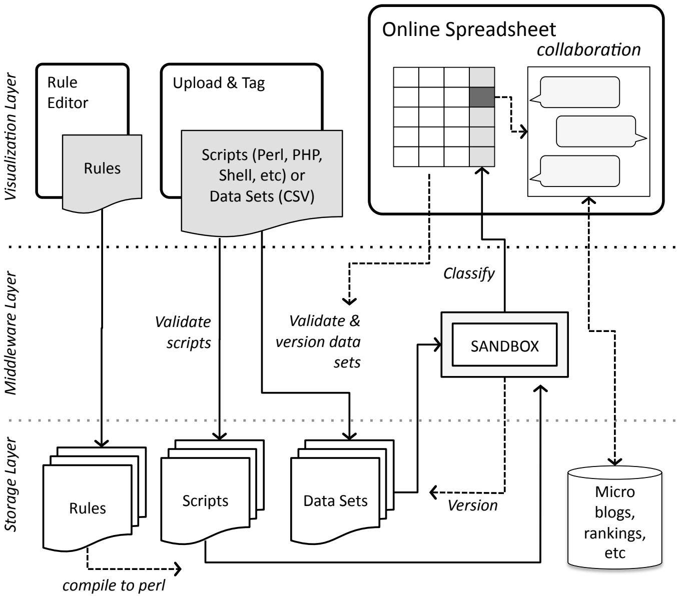


Figure 3. High-level overview and information flow of xPad

the scope of the inclusive rules, without introducing the logical NOT (in the interest of keeping the rules simple for the users). Thus, the exclusive rules are used to lower the coverage and increase the accuracy of the rules. Generally, the users are striking a balance between coverage and accuracy, with more preference on accuracy.

For better rule management we organize the rules according to the failure code (class). This helps the user to focus on a subset of rules, evaluate their performance, and make rule update and edit. Table 1 illustrates a rule-based classifier called *FC-TOP10*, where TP indicates the rule matches a ticket and generates the correct classification, FN indicates the rule fails to match a ticket, and FP indicates the rule matches a ticket and generates an incorrect classification. The rules are organized per failure code (e.g., there are three rules for failure code CAP01 - File system full). The rule statistics are also provided at the classifier level, the rule group level (per failure code), and the rule level. Note that the coverage rate at the latter two levels is defined relative to the number of tickets belonging to the failure code. Although it may appear strange to see a coverage rate larger than 100%, it helps understand and update the individual rules.

4 xPad Crowdsourcing Platform

In this section we describe our implementation of the solution presented in Section 3, an *xPad* crowdsourcing platform that supports rule-based classification by leveraging the mass collaboration enabled by Web 2.0 technologies. Our approach combines rule-based classification with crowdsourcing to leverage the network of user communities and domain expertise and provide an easy means to update and validate the rule base.

xPad is implemented on a LAMP (Linux, Apache, MySQL, and PHP) stack with heavy use of AJAX (Asynchronous JavaScript and XML) to give users an interactive experience. xPad is designed to enable a large user community to both author (or create) and consume (or run) rules against community uploaded data sets. There are two types of users that are of interest: rule authors and rule consumers, with consumers executing the authored rules against uploaded data sets. Both type of users are expected to use xPad as a collaborative environment for improving rule efficiency and sharing classified data sets.

Figure 3 shows a high-level overview of xPad, highlighting how information flows through the system. Looking at the life-cycle of classifying a data set, there are five distinct stages: (1_a) script or rule authoring, (1_b) data upload, (2_a) automatic classification, (2_b) manual override, and (3) data export. Here, X_a and X_b denotes two stages that can happen in parallel by different users. Stages 1_a and 1_b are performed in the *Rule Editor* and *Upload and Tag* in Figure 3, respectively. As we will describe shortly, users can also upload executable classification scripts directly. Stages 2_a, 2_b, and 3 are performed in the *Online Spreadsheet*. Finally, in all stages, xPad supports blog-like collaboration on user activities.

That said, there are three primary components in xPad: (1) an online spreadsheet editor, (2) a script sand-box, and (3) a collaboration engine. The first is best described as a simple interactive online spreadsheet. Users can upload their data (shown in *Upload and Tag* in Figure 3). Once uploaded, they are able to manually modify or apply user-uploaded scripts or rules.

The second component is the script execution sand-box. xPad allows users either to create classification rules using the *Rule Editor* or to upload executable scripts authored using a variety of scripting languages, including Perl, PHP, and shell script. Internally, xPad translates all rules to Perl scripts. The structure of each script is fairly simple: it must read CSV data from standard input and writes to standard output. Each line in the output is a three-tuple: (affected row,classification code,confidence). Also, only affected rows are expected to be piped to output.

Because xPad allows users to upload arbitrary executable scripts, these scripts need to run in an isolated environment to protect the security and integrity of the system (even if users are not expected to author malicious code). That said, before any script is executed, xPad creates an ephemeral sand-box, copies the necessary data and executable files, and runs the scripts against the data. The output is then piped back to the user visualization logic, which in turn updates the displayed table on the user's browser.

To minimize the effect of user errors, xPad automatically versions all data manipulation, particularly before script execution. Here, we wanted to give users the ability to role back any unintended changes. Additionally, it allows easy comparisons between the efficacy of different scripts. Going forward, we see the use of sand-boxing diminishing as users will rely more on the rule editor, which internally ensures that the rules are translated to safe scripts.

The last component of xPad is an integrated collaboration engine. Here, xPad allows users to add discussion threads against all user-created contents, including scripts, data sets, etc. Each thread is tagged and can be filtered across various dimensions.

5 Experimental Evaluation

In this section we illustrate how the rule-based crowdsourcing classification approach can be used for problem ticket classification, and also compare it with the supervised learning approach.

Our evaluation is based on problem ticket classification data collected from several large service delivery centers. The data are collected over a period of one year, including more than 20,000 problem tickets. However, the data are not uniformly distributed across delivery centers; some newer centers only have several months of data.

5.1 Evaluation Results

Table 2 shows the classification results based on the data from one delivery center. Only the largest 10 classes are shown in the table. We use one week of data (500 tickets) as the training data to create the rules, and another four weeks of data (2158 tickets) for testing. We use TP to denote true positives (the number of tickets correctly classified by rules for their classes), FN to denote false negatives (the number of tickets not classified by rules for their classes), and FP to denote false positives (the number of tickets incorrectly classified by rules for other classes).

Table 2. Rule-based classification results. (TP: true positives, FN: false negatives, FP: false positives.)

Failure Code	Training				Testing			
	Total	TP	FN	FP	Total	TP	FN	FP
File system close to full	97	94	3	7	317	287	30	42
CPU/memory close to full	99	96	3	2	340	302	38	56
Unaccessible /not pinging	30	25	5	13	113	74	39	51
Disk space	7	4	3	3	20	10	10	8
Service / daemon down	5	5	0	8	66	34	32	21
OS issues	49	48	1	0	63	48	15	1
Password reset / unlock	7	5	2	0	53	47	6	5
ID creation	24	13	11	3	111	72	39	22
ID deletion	17	16	1	5	112	104	8	20
Reboot request	10	8	2	3	54	38	16	12
Total	345	314	31	44	1249	1016	223	238
Coverage	72%				58%			
Accuracy	88%				81%			

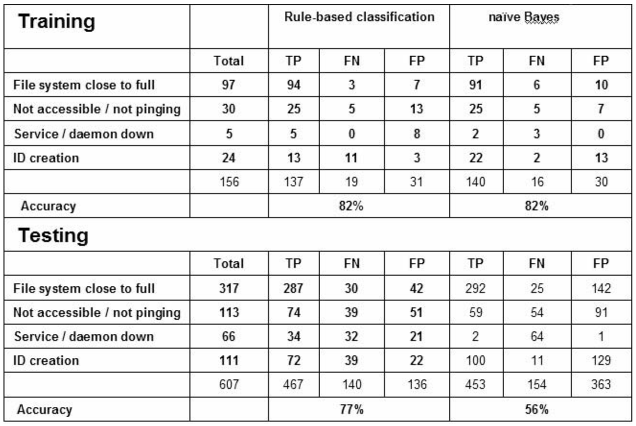
Note that the above counts are calculated on a per class basis. Since one ticket may result in rule matches from several different classes, the overall coverage and accuracy metrics calculated based on the sum of the above counts (e.g., the accuracy of the training set is calculated as 314 / 358 = 88%) may be different from the actual metrics. However, we found the difference is not significant, and thus use these aggregated metrics as an approximation of the overall classifier performance.

5.2 Comparison with Supervised Learning

We compare the rule-based classification with several learning algorithms such as binary decision tree and naïve Bayes. In the interest of space, in this paper we only show the results from naïve Bayes, which had better accuracy compared to the other learning approaches we tested on this data set.

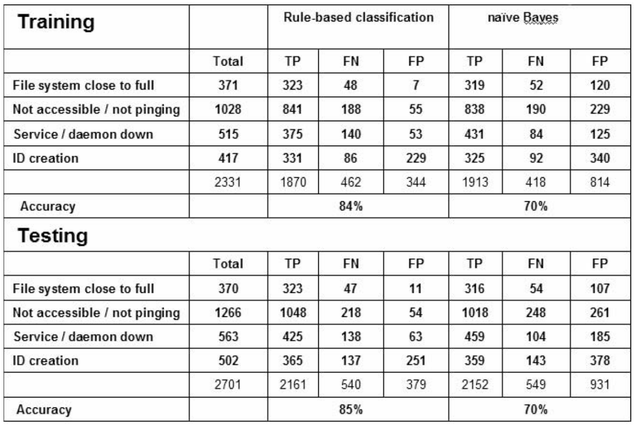
Generally, supervised learning techniques perform well in terms of accuracy, if enough labeled data are available. We have found, however, that this assumption is not fully satisfied when the cost of labeling data is prohibitively large. For example, Table 3 is a comparison based on a smaller training set (500 tickets), both the rule-based classifier and the naïve Bayes approach have about 80% of accu-

Table 3. Comparing rule-based classification with naïve Bayes - smaller data set. (TP: true positives, FN: false negatives, FP: false positives.)



Training		Rule-based classification			naïve Bayes		
	Total	TP	FN	FP	TP	FN	FP
File system close to full	97	94	3	7	91	6	10
Not accessible / not pinging	30	25	5	13	25	5	7
Service / daemon down	5	5	0	8	2	3	0
ID creation	24	13	11	3	22	2	13
	156	137	19	31	140	16	30
Accuracy		82%			82%		
Testing							
	Total	TP	FN	FP	TP	FN	FP
File system close to full	317	287	30	42	292	25	142
Not accessible / not pinging	113	74	39	51	59	54	91
Service / daemon down	66	34	32	21	2	64	1
ID creation	111	72	39	22	100	11	129
	607	467	140	136	453	154	363
Accuracy		77%			56%		

Table 4. Comparing rule-based classification with naïve Bayes - larger data set. (TP: true positives, FN: false negatives, FP: false positives.)



Training		Rule-based classification			naïve Bayes		
	Total	TP	FN	FP	TP	FN	FP
File system close to full	371	323	48	7	319	52	120
Not accessible / not pinging	1028	841	188	55	838	190	229
Service / daemon down	515	375	140	53	431	84	125
ID creation	417	331	86	229	325	92	340
	2331	1870	462	344	1913	418	814
Accuracy		84%			70%		
Testing							
	Total	TP	FN	FP	TP	FN	FP
File system close to full	370	323	47	11	316	54	107
Not accessible / not pinging	1266	1048	218	54	1018	248	261
Service / daemon down	563	425	138	63	459	104	185
ID creation	502	365	137	251	359	143	378
	2701	2161	540	379	2152	549	931
Accuracy		85%			70%		

racy in training, but the accuracy of naïve Bayes deteriorates to 56% in testing while the rule-based classifier sustains at 77%.

On the other hand, when a large amount of pre-classified data are available, the supervised learning may also have challenges in featurization and capturing the right phrases, especially for machine-generated tickets. This is the case as shown in Table 4. Although the classification performance is consistent between training and testing for both the rule-based classifier and naïve Bayes, the naïve Bayes approach tends to have a higher number of false positives.

5.3 Cost Analysis

While the rule-based classifier has shown promising accuracy results, it it also important to understand the complexity and cost of creating and managing the rules. We use the number of patterns (either words or phrases) rather than the number of rules to quantify the classifier complexity, and consider the time spent on getting these patterns as the cost.

In our exercise, the first set of rules built for the largest 10 failure code classes, as shown in Table 2, has about 100 patterns (i.e., 10 patterns per failure code). Building this rule set took about 5 hours, that is, 0.5 hour per failure code class. Although spending more time can help to increase the rule coverage and accuracy, at certain point, it will be more productive to manually classifying the tickets instead of finding the obscure and not repeatable patterns.

As we progress, the rule set grows from covering 10 failure codes to 29 failure codes and work has also been conducted to update the rules for the first 10 failure codes in response to ticket mix changes. The new rule set has 500 patterns (i.e., 17 patterns per failure code). This increases the coverage from 60% to 70%, and still keep the accuracy around 80%. The time spent is 1.5 hour per failure code on average. Overall, we believe the cost of creating the rule set is manageable, compared to the cost of labeling the tickets. It also has the value of sharing the knowledge and rules across the delivery centers to realize collaborative savings.

6 Conclusions and Future Work

Problem management is a critical and expensive element for delivering IT service management and touches various levels of managed IT infrastructure. While problem management has been mostly reactive, recent work is studying how to leverage large problem ticket information from similar IT infrastructures to probatively predict the on sit of problems.

Because of the shear size and complexity of problem tickets, supervised learning algorithms have been the method of choice for problem ticket classification, relying on labeled (or pre-classified) tickets from one managed infrastructure to automatically create signatures for similar infrastructures. However, where there are insufficient pre-classified data, leveraging human expertise to develop classification rules can be more efficient. In this paper, we describe a rule-based crowdsourcing approach, where experts can author classification rules and a social networking-based platform (called *xPad*) is used to socialize and execute these rules by large practitioner communities. We also provide a detailed study of the efficacy of rule-based classification, across two dimensions: accuracy and cost. We show that well constructed (simple) rules can perform better than supervised learning, and is cost effective as well.

There are several directions for future work. The most interesting one is to study how to refine the rule editing process by incorporating machine learning. The method presented in this paper does not explore methods combining machine learning with knowledge acquisition. However, there is a substantial body of work in this combination as applied to various domains ([14], [17], [18], among many others). Finding the "sweet spot" minimizing effort by experts on both of the sources of expert knowledge while maintaining a given level of accuracy requires some effort and is domain-specific; however, the domain of problem tickets is large enough and has great enough business value that it worth the effort.

In addition, it is also interesting to develop a methodology to streamline the process of identifying patterns in a consistent and efficient way in response to new client onboarding or failure code changes. Furthermore, it is worthy to study whether exploring the hierarchical structure of the failure code can help to improve the accuracy of problem ticket classification.

Acknowledgement

The authors would like to thank Jason Gast for many useful comments and suggestions on understanding the problem management process and acquiring the knowledge for ticket classification.

References

[1] "IT Infrastructure Library. ITIL Service Support, version 2.3." Office of Government Commerce, June 2000.

[2] A. Bose, A. R. Heching, and S. Sahu, "A framework for model based continuous improvement of global it service delivery operations," in *Proceedings of IEEE International Conference on Services Computing, Honolulu, HI*, pp. 197–204, 2008.

[3] B. Yu, "An evaluation of text classification methods for literary study," *Literary and Linguistic Computing*, vol. 23, pp. 327–343, 2008.

[4] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng, "Some effective techniques for naive bayes text classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 1457–1466, 2006.

[5] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, pp. 513–523, 1988.

[6] T. Joachims, *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers / Springer, 2002.

[7] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[8] W. Cohen and H. Hirsh, "Joins that generalize: Text classification using WHIRL," 1998.

[9] C. Apté, F. Damerau, and S. Weiss, "Automated learning of decision rules for text categorization," *ACM Transactions on Information Systems*, vol. 12, pp. 233–251, 1994.

[10] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[11] M. E. Ruiz and P. Srinivasan, "Hierarchical text categorization using neural networks," *Inf. Retr.*, vol. 5, pp. 87–118, January 2002.

[12] R. Michalski and R. Chilausky, "Knowledge acquisition by encoding expert rules versus computer induction from examples: a case study involving soybean pathology," *International Journal of Human-Computer Studies*, vol. 51, pp. 239–263, 1999.

[13] A. Ben-David and E. Frank, "Accuracy of machine learning models versus "hand crafted" expert systems - a credit scoring case study," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 5264–5271, 2009.

[14] T. Mitchell, S. Mahadevan, and L. Steinberg, "LEAP: A learning apprentice for VLSI design," 1985.

[15] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second ed., 2005.

[16] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[17] D. Loewenstern, S. Ma, and A. Salahshour, "PICCIL: Interactive learning to support log file categorization," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, (Washington, DC, USA), pp. 311–312, IEEE Computer Society, 2005.

[18] G. Towell, J. Shavlik, and M. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks," 1990.