Pragmatic Text Mining: Minimizing Human Effort to Quantify Many Issues in Call Logs

George Forman
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304
ghforman@hpl.hp.com

Evan Kirshenbaum
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304
kirshenbaum@hpl.hp.com

Jaap Suermondt
Hewlett-Packard Labs
1501 Page Mill Rd.
Palo Alto, CA 94304
suermondt@hpl.hp.com

ABSTRACT

We discuss our experiences in analyzing customer-support issues from the unstructured free-text fields of technical-support call logs. The identification of frequent issues and their accurate quantification is essential in order to track aggregate costs broken down by issue type, to appropriately target engineering resources, and to provide the best diagnosis, support and documentation for most common issues. We present a new set of techniques for doing this efficiently on an industrial scale, without requiring manual coding of calls in the call center. Our approach involves (1) a new text clustering method to identify common and emerging issues; (2) a method to rapidly train large numbers of categorizers in a practical, interactive manner; and (3) a method to accurately quantify categories, even in the face of inaccurate classifications and training sets that necessarily cannot match the class distribution of each new month's data. We present our methodology and a tool we developed and deployed that uses these methods for tracking ongoing support issues and discovering emerging issues at HP.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Types of Systems—*decision support;* I.2.6 [**Artificial Intelligence**]: Learning—*concept learning;* H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*abstracting methods, linguistic processing;* I.5.4 [**Pattern Recognition**]: Applications—*text processing;*I.5.3 [**Pattern Recognition**]: Clustering—*algorithms.*

General Terms

Algorithms, Measurement, Design, Economics.

Keywords

text mining, log processing, supervised machine learning, quantification, text classification, applications.

1. INTRODUCTION

High-quality customer service and support is a high priority for Hewlett-Packard. This affects the company's reputation and customer satisfaction directly. There is a persistent need to *identify* and *quantify* the most frequent issues for every product. For a large enterprise having many product lines of high technology products like HP, this is a complex task. The resulting information is used in a variety of ways: to ensure that easy help (e.g. automated diagnostics, patches, and documentation) is available for the most frequent problems; to efficiently target engineering resources at fixing problems or even redesigning products; to track whether we are reducing these issues (and the costs associated with them); and to compare to industry averages.

A prolific and valuable source of information about these issues is contained in the records from call centers. Each incident results in a description of what caused the customer to call and what was done to resolve the issue. Unfortunately, these call records are in (terse) free text, because the technicians handling calls are focused on (and rewarded for) resolving incidents rather than spending their time describing them well. As a result, it is difficult to use the information that is in the call-record text.

In this paper, we describe a solution we have developed that harvests the information buried in these hundreds of thousands of call records. Our solution enables a large amount of automation in reporting issues with a relatively small amount of up-front work.

The *automation* consists of taking a batch of cases (such as a month's worth of calls for a particular product line of interest) and feeding them to a trained automated quantifier, which returns the distribution of the cases over the issues. The results are then fed into a reporting tool, which generates the appropriate graphs, month-to-month comparisons, alerts, reports, etc.

The *up-front work* is where the key data mining contributions are, particularly in the method and tool we created to make it practical to very quickly create an accurate quantifier for a new problem domain (e.g., for each of hundreds of product lines). Typically, to start with, all we have is a batch of raw, unlabeled cases. We need to find what issues are prevalent in the cases, and build up a labeled training set of cases for these issues. Our core contribution is a tool and methodology that allows one to do so very efficiently. This tool involves clustering, classification and quantification. From a data mining perspective, we solved some key problems in novel ways: (1) we developed a new clustering method to identify meaningful topics in large bodies of (dirty) text [16]; (2) we worked out an approach to train a classifier quickly, robustly, and intuitively, even one that involves a large number of categories; and (3) we developed methods to accurately quantify categories even in the face of inaccurate classifications [5][6].

The remainder of this paper is organized as follows. First, as background, we describe existing approaches in industry, and we show a sample of data to ground our approach. We then describe our methods and the end-to-end solution. We present the technology transfer from the research labs to our business units and the results of deployment. Finally, we discuss lessons learned, open issues, and future work.

1.1 Existing Approaches

The first question we usually get when we talk about our work in automated problem quantification is "Wait… don't we do this already?" The short answer is: yes, lots of companies do it, but nobody does it very well, nor do they do it efficiently.

Historically, companies have based their tracking of warranty and support issues on *parts data*: If my support business replaces part X very often, maybe I have a quality problem there. Parts data can give us detailed and accurate data about quality problems, but it has limitations. Most importantly, it does not tell us anything about a vast number of calls that offer low-hanging fruit from a cost-reduction perspective: calls that could have been prevented and calls that can be resolved on-line or over the phone– customers asking questions, set-up problems, configuration problems, patchable problems, documentation needs, and so on. Furthermore, parts data do not give us any direct information about customer satisfaction issues, nor details of what customers are asking about or asking for.

The other common approach is to make the call center decide on a code for every call, and for the analysis to use the resulting codes. Typically, there is a deep hierarchy of possible codes for any product. This provides computer-readable information and allows reporting on the frequency with which various codes were used for different products. Various tools exist for generating interesting graphs and reports, drill-down, trends, etc. Unfortunately, this approach has several critical weaknesses. First, the codes used are inaccurate. We have run several internal studies that document that there is poor correlation between the code assigned to a call and the actual problem as reflected in the record or transcript of the call. This is not surprising, given that the call agents are required to choose a code but are not incented to pick a *good* code, resulting of in a preponderance of cases labeled with "misc problem" as well as codes near the top of the menu. Second, using codes is expensive. The hierarchies need to be created and maintained, the call agents need to be trained in the codes, and then on every call they need to spend precious time selecting a code from a potentially very large hierarchy. Third, this misses all new or emerging problems, because codes do not yet exist or the call agents have not yet been trained in recognizing them.

In addition, there are prevalent ad-hoc approaches: eye-balling the data; looking at FAQ lists and trying to notice any important missing items; periodic sampling and manual labeling; expert opinion; availability of expertise; etc.

An appealing alternative, particularly from the perspective of a data mining researcher, is to use the raw text in the customer-support call logs to quantify the occurrence of problems. Unfortunately, well-known data mining techniques are not in frequent use in this domain because there are numerous barriers that prevent them from being effective, besides the usual challenges that affect text mining even in "clean" domains such as news stories.

Word counts—the simplest form of text-based quantification—suffers from extensive noise from domain-specific stop words, as well as from the fact that many keywords are related to the same issue (e.g., "battery", "charge", "power", "weak").

Text clustering is appealing because of its apparent lack of need for human effort. In our experience and experimentation with known text clustering techniques on our support data, however, we have found them to often yield results that can be described as worthless. Unrelated problems are merged; there are many meaningless clusters; single issues are split into many small clusters because of nuances in the language used and the varying ways that different speakers describe the same thing. On top of that, one of our key needs is month-to-month tracking, which is hard if each month's clusters are different and not easily linkable. The recent work by Mei and Zhai [13] and the somewhat older ThemeRiver visualization by Havre et al. [9] both attempt to detect the emergence of themes and track their trends through time in a text stream via clustering. These methods focus on only the *largest* themes, and the cluster boundaries are influenced strongly by the competing clusters, rather than the semantic edge of an important issue, yielding uncalibrated quantities. By comparison, *supervised* machine learning can focus on less frequent but nevertheless important classes, and the quantification it produces can be used to accurately measure the cost associated with each issue [5][6].

The appeal of classification is its reliability, performance and repeatability given a particular set of problems and a good training set. The key drawback has been the large effort required for training categorizers. Historically, people have used keyword-based rules in this domain, which have a high up-front modeling effort, debatable accuracy, and a substantial maintenance cost. Alternatively, categorizers based on machine-learning require labeled training data, typically on the order of 100 cases per category. This requirement quickly becomes prohibitive when we are talking about hundreds of categories per product, for hundreds of products; even defining and maintaining the categories stretches the amount of effort humans are willing to make.

Quantification is a problem that has received very little attention in the data mining research community despite the tremendous practical need for it in many applications. When we first tackled this problem, we had quite a bit of experience in mining customer-support data, including categorization and clustering of solution documents and call records. We thought this would be a straightforward application of categorization (classify all items and simply count positive predictions), possibly augmented with some text cleaning and feature selection. But an interesting problem in quantification, at least from the perspective of periodic reporting, is that you fundamentally need to *assume* shifts in the class priors: If the class distribution does not change, you don't need an automated quantifier: you only need to manually quantify the first batch. As a result of this time-varying behavior that is inherent to the application, the obvious solution above—"classify-and-count"—does not work well, as the usual machine learning classification methods assume that the test set and training set are drawn from the same distribution [4].

1.2 Sample Call Log Data

To give the reader an idea of what we are up against, Table 1 shows a sample of text notes recorded by technical support agents as they serve customers on the phone. This sample is taken from the HP iPAQ handheld product line, but is representative of similar call notes for hundreds of other product lines, and likely analogous to those experienced in many companies: the quality of spelling, grammar and completeness about the subject is much poorer than in common text research corpora such as news articles. In more complicated product lines, such as servers, the call notes run much longer about the diagnosis and resolution processes. Nonetheless, they too are terse and the context of the problem is often assumed, not recorded. Depending on the product line, the call volume can provide on the order of hundreds of thousands of calls per month in a company the size of HP.

Because of the nature of technical support, a large fraction of calls are about one-of problems. Depending on the complexity of the product line, easily half the cases do not lead to interesting clusters of issues. Hence, simply reading a random sample is inefficient and only works to determine the top few issues. In contrast, our typical users prefer to distinguish 10-100 issues.

2. METHODOLOGY

We now describe in more detail our method and tool for making it practical to quickly create many accurate quantifiers. We provide a high-level overview here, and focus on the research contributions in more detail in Section 3.

First, we decide on the domain for the analysis. The domain is typically a particular product line, such as the iPAQ handhelds in the data shown in Table 1, and is usually further limited to cases in the language the analyst speaks.

Next, we collect raw call data for the domain. Call data typically includes (at a minimum) a product identifier or model number. If such an identifier is not specified as structured data, it can typically be derived reliably from serial numbers or contract data, or otherwise is usually somewhere in the body of the call record in text form. The product identifier allows us to filter the call data to at least some approximation of the domain of interest, so that we don't try to identify iPAQ problems in calls about storage arrays.

The next step is to generate ideas for potential categories. To do so, we run our clustering tool, the "Ten-Second Answer," on the cases and display the results. These include cluster descriptions (potential category names) as well as cluster size estimates and a sample of cases. We describe the clustering tool and the underlying method in more detail in Section 3.1.

Next, a human domain editor starts to create categories and identify examples for each category. We describe this process in detail in Section 3.2. This process, which we call "search and confirm," turns classical training upside down. The domain editor takes a category of interest, searches for possible examples, scans the results and confirms whether the cases are indeed examples of the category.

The training examples are used in real time, in the background, to train a classifier for each category. Initially, the classifiers will be poor and biased, but they refine quickly as data are added. Based on the results of classification, we run our quantification method in real time to provide estimates of the size of each category (described in more detail in Section 3.3 and in [5][6]). The results are displayed and updated in real time to provide the user with feedback.

Table 1. Sample cases from call logs on the HP iPAQ.

Time	Model	Subject
12:24	3955	H3955 how to hard reset the ipaq
12:26	3955	H3955 Cannot use email on wifi
12:33	3955	H3955 Cannot sync his Outlook items with his
12:36	3765	3765 port on ipaq not makeing good connection
12:40	3835	iPAQ H3850 -Display /white screen
12:45	3835	h3630-eu instal printer and now cannot connect
12:45	1910	Sounds and notifications
12:45	3955	3955-no outlook choices to sych
12:46	1910	H1910 - Wants to know where the reference guide
12:46	3955	H3950: Unit is giving memory full errors
12:47	3850	h3850-battery is not holding chargeand tapped on
12:48	3650	eu ask if ipaq has adjusted his warranty status
12:54	3630	H3630 - The ipaq ill only turn on when it is on the
12:54	3835	3835 cracked case and ip SR itself
12:54	1910	H1910: unit has a cracked screen;screen taps don't
12:59	3850	Backlight will not turn on
13:01	1910	H1910 - Wanted to know how to restore.
13:07	3670	H3670: Unit is not synching with laptop
13:13	3955	communications error 607
13:13	3630	3650 screen is corupted
13:20	3835	Getting Shell32.exe errors
13:22	3850	H3850-cannot pass the logo screen

The domain editor can graphically specify various types of relationships among categories—for example, multiple independent hierarchical relationships, mutual exclusivity among certain topics, allowable overlap, etc. Thus, a flexible topic hierarchy for the domain is created interactively. Alternatively, a pre-existing topic hierarchy can be specified or loaded.

We have included several capabilities for helping to determine whether the quantifier is complete (for example, whether there are remaining latent categories). One such is residual analysis. In residual analysis, we query for cases that have a low probability (by the current classifiers) of being in any particular category. We then cluster these cases to see whether there are categories we missed. A related technique is topic drill-down: we grab cases that are already labeled or predicted to be in a particular category, and we cluster those to look for subtopics. Also included are various performance measures via cross-validation.

When the domain editor decides the categorizers and quantifiers are good enough, he or she can export them for off-line use (for example, on data from subsequent months). We also have an associated reporting tool, which allows generation and publication of graphical reports on a web site for interactive exploration by the many stakeholders and information consumers.

3. CONTRIBUTIONS

3.1 Text Clustering for Issue Identification

The first task is to identify issues that commonly lead to calls based on the textual descriptions in the call logs. The hope was that some sort of clustering method would allow the identification of at least the most important issues in a fully-automated way. Unfortunately, traditional clustering methods tend not to work well on this sort of data. Typically, such methods treat the presence or absence of individual words as a high-dimensional space and attempt to find points in that space that partition the cases. (Some clustering algorithms, e.g. [1], can return overlapping clusters.) Empirically, this tends not to find useful clusters in the sort of data in call logs, especially when the number of potential clusters becomes large. It has the further drawback that the desired output is not the clustering per se (*i.e.*, the particular cases assigned to each cluster) but a human-understandable *description* of each cluster to help the user identify useful themes or categories. Once a clustering has been done, it remains to analyze the case descriptions to generate a summary of what each cluster represents.

We also require a method that is fast enough for interactive use on large datasets, ruling out methods that require factoring large matrices (e.g. latent semantic indexing [3]) or many iterations for convergence (e.g. EM, k-means) [e.g. 1,12]. Beil et al. recently presented a fast method for text clustering, which reduces the dimensionality of the feature space to that of frequent term sets [2]. They report a runtime of a few minutes for several thousand cases, but with a quadratic trend. As described in the next section, our algorithm is used interactively to find clusters on the results of searches, and for this it is important that it be able to cluster 10,000–100,000 cases in a matter of seconds so as to not disrupt the user's train of thought

To address our needs in this project, we developed a new clustering algorithm that focuses on identifying useful, inherently describable clusters from text rather than on assigning individual cases to clusters. As such, it makes no attempt to be either mutually exclusive or exhaustive, reflecting the facts that (1) not infrequently, a call may be the result of more than one issue, (2) any discovered set of topics will almost certainly be incomplete, and forcing the system to fit every call into a cluster just degrades the usefulness of the clusters, and (3) some case descriptions are simply too impoverished to be useful and might as well be ignored. The algorithm automatically organizes the clusters found into a hierarchy. This proves useful in our domain; for example, there are multiple subtypes of software problems and hardware problems. The algorithm does not require the user to provide a target number of clusters or hierarchy depth—these are inferred from the data. Finally, since the clusters found are defined by commonalities between words, it is straightforward to create easily understandable textual descriptions of the clusters.

The algorithm begins by extracting words from the case descriptions, canonicalizing them (regularizing case, stemming, removing stopwords), and then counting the number of cases each word and each pair of words appears in. This is used to construct a matrix in which the rows and columns are words and in which each element is the signed bi-normal separation [8] value of one word as a predictor of the other. From this matrix, each pair of words is calculated to be *positively correlated* (at least one ordering has a sufficiently positive value in the matrix and the reverse ordering does not have a sufficiently negative value), *negatively correlated* (the reverse), or *uncorrelated*.

A key insight is that many topics consist of multi-word phrases, the words of which tend to appear together and are therefore positive correlated. On the other hand, competing "sibling" topics tend to have words that are negatively correlated. These competing topics can share one or more words, reflecting a "parent" topic. For example, if "mirror" and "broken" are positively correlated, it indicates that broken mirrors may be a cluster, while if "mirror" and "rear" and "mirror" and "side" are positively correlated, while "rear" and "side" are negatively correlated, it indicates that broken rear-view mirrors and broken side mirrors are likely subclusters of the broken mirror cluster.

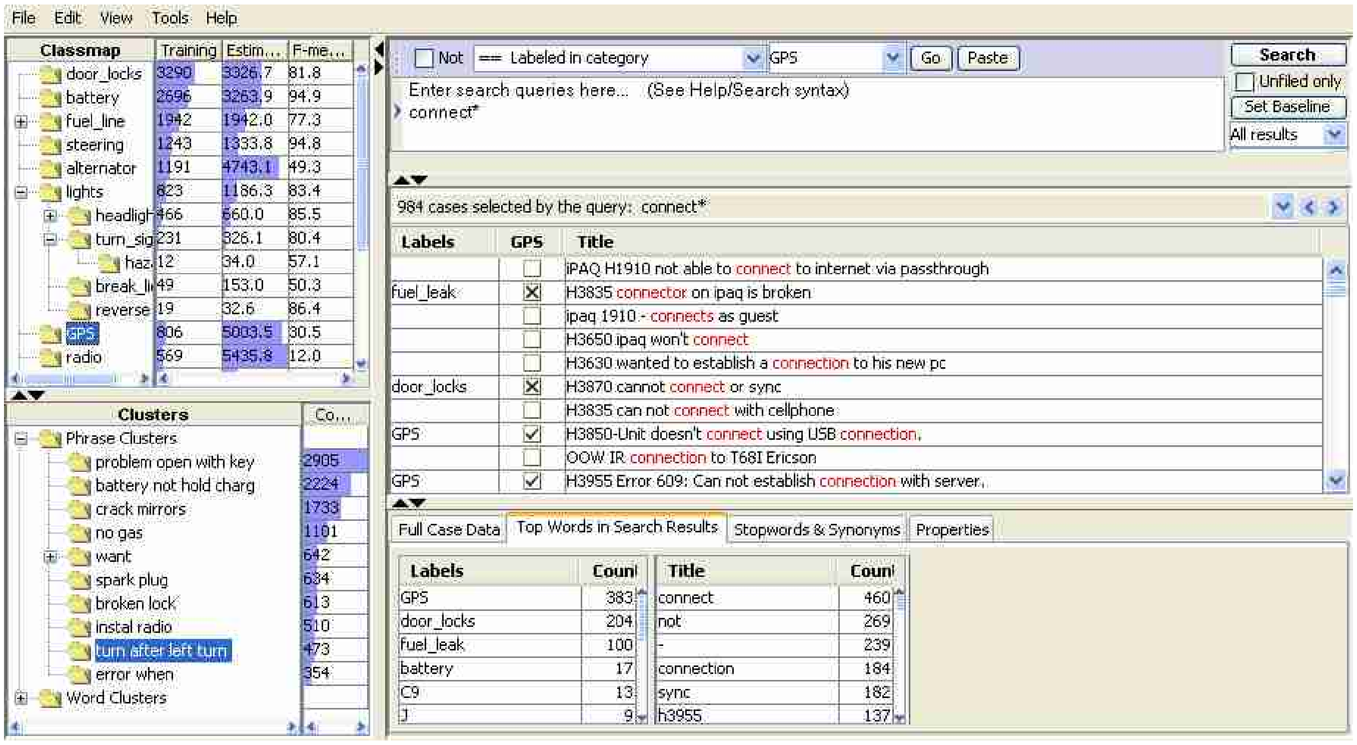
For the details of the algorithm, we must refer readers to a forthcoming paper [16]. Here we can only briefly describe how it works: First, the most frequent pairs of words that are positively correlated are examined in turn to create maximal sets such that no pair of elements in a set is negatively correlated. The resulting sets are the cluster definitions. The next step is to place the clusters into a hierarchy. This is accomplished by recursively partitioning the clusters based on the negatively-correlated pairs. Next, cluster descriptions are derived from the words that make up the cluster sets. The partition branches are described by the negatively-correlated terms that induce them as well as words that are correlated with them. The generalizations are described by the words common to all branches. The correlation and most common relative order of the terms in the case descriptions can be used to determine an ordering of the words that makes the description more readable. For example, if both "charge" and "hold" are seen to be descriptive of a cluster, the description will likely contain "hold charge" rather than "charge hold" based on common usage. Finally, based on the presence of words from the cluster sets in the actual cases, we select representative examples of each cluster and we provide a rough estimate of the cluster size.

The resulting analyses were well received and the prototype implementation was dubbed the "Ten Second Answer" for its ability to produce useful results quickly and without any human intervention. It became apparent, however, that this was not a complete solution. While it found many "real" clusters, it would miss some that were obvious and it would latch on to groups of words that didn't describe issues, but were rather words that tended to go together (like "customer wants") in the description of many calls. More importantly, there was no notion of *stability* of clusters from one month to the next, and it was hard to get a reliable estimate of the number of calls associated with each cluster. So, it was not straightforward to be able to identify the most important clusters or to tell whether the issues they represented were growing or shrinking.

Despite these problems, the clusters found—at least the ones that obviously described issues—did give users a good idea about at least some of the problems that were there, so we next turned our attention to seeing whether we could leverage this, along with our previous work in classification, into a "One Hour Answer", a tool that would require a bit of work on the part of users but that would be straightforward to use and give the type of supervised learning results that were actually required.

3.2 Method for Constructing and Training Many Categorizers, with Hierarchy Discovery

Although fully-automated clustering is insufficient, it is often useful for the next stage in our process, constructing a hierarchy of issue categories and training categorizers to recognize issues of each. Unlike typical approaches, we treat these two activities as a single step in which the hierarchy is constructed as a byproduct of the same computer-assisted process that results in the trained categorizers.



Classmap	Training	Estim...	F-me...
door_locks	3290	3326.7	81.8
battery	2696	3263.9	94.9
fuel_line	1942	1942.0	77.3
steering	1243	1333.8	94.8
alternator	1191	4743.1	49.3
lights	823	1186.3	83.4
headligh	466	660.0	85.5
turn_sig	231	326.1	80.4
haz.	12	34.0	57.1
break_li	49	153.0	50.3
reverse	19	32.6	86.4
GPS	806	5003.5	30.5
radio	569	5435.8	12.0

Clusters	Co...
Phrase Clusters	
problem open with key	2905
battery not hold charg	2224
crack mirrors	1733
no gas	1101
want	642
spark plug	634
broken lock	613
instal radio	510
turn after left turn	473
error when	354
Word Clusters	

Enter search queries here... (See Help/Search syntax)
connect*

984 cases selected by the query: connect*

Labels	GPS	Title
	☐	iPAQ H1910 not able to connect to internet via passthrough
fuel_leak	☒	H3835 connector on ipaq is broken
	☐	ipaq 1910 - connects as guest
	☐	H3650 ipaq won't connect
	☐	H3630 wanted to establish a connection to his new pc
door_locks	☒	H3870 cannot connect or sync
	☐	H3835 can not connect with cellphone
GPS	✓	H3850-Unit doesn't connect using USB connection.
	☐	OOW IR connection to T68I Ericson
GPS	✓	H3955 Error 609: Can not establish connection with server.

Full Case Data | Top Words in Search Results | Stopwords & Synonyms | Properties

Labels	Count
GPS	383
door_locks	204
fuel_leak	100
battery	17
C9	13
J	9

Title	Count
connect	460
not	269
-	239
connection	184
sync	182
h3955	137

Figure 1. Analysis tool. (Actual issue names and clusters have been blinded.)

Traditional approaches to training categorizers are difficult for the human experts who are asked to provide the judgments that result in the labeled training sets. The expert will be presented with a case, selected at random from a set of unlabeled cases and asked "Which of these fifty issues (if any) is the correct issue for this case?" or else, focusing on a single issue, asked "Is this the correct issue for this case?"

The former question is one that appears to be cognitively relatively difficult for people to answer, especially when the number of possible issues gets much above about four or five, and, consequently, each case takes a non-trivial amount of time to label. (In real customer-support systems, dozens of categories are the norm.) Also, as people's expertise varies, some people may be able to accurately assign cases to only some of the issues, resulting in either inaccurate guessing on the other issues or a need to have multiple experts go through the cases.

The latter question seems to be much easier for people, and it allows different experts to focus on different issues in their areas of expertise, but it requires a pass through the cases for each issue, which is extremely time-consuming.

Another problem that both approaches suffer from is that when an issue is rare, a large number of cases may need to be seen before a sufficient number of instances of that issue have been labeled for the categorizer associated with the issue to be reliable. If the learning method requires, say, fifty positive examples, but an issue is responsible for only 1% of the cases in the unlabeled set, an expert would expect to be presented with 5,000 cases, each to be judged individually, before a reliable categorizer could be trained.

In our approach, building training sets and identifying issues is transformed into an interactive process that calls upon the human expert to perform several tasks that are each simple for people.

The user is presented with a display like Figure 1. At the bottom left are the results of the fully-automated clustering over the cases. At the bottom right is an analysis of the most frequently appearing words and phrases in the descriptions associated with the cases. At the top left is the current hierarchy of issue categories (called the "classmap," which may contain multiple hierarchies). Initially, for a new domain, this will be empty, but if one is previously established, it can be used as a starting point. And at the top right is a random sample of cases.

The user's first task is to identify one or two issue categories. Because the hierarchy can be changed at any time without sacrificing invested effort, it is not important that these be at the "right" level of generality (neither too general nor too specific), nor that they even appear in the final hierarchy. They need not be the most common issues nor the most important. Given the sample cases, the clustering analysis, and the frequent words and phrases, it is almost impossible for a person to *not* identify one or two good candidates: "There seem to be a lot of cases involving screen problems" or "Batteries seem to be a problem."

Once these initial categories have been added to the hierarchy, the user focuses on one of them and finds positive examples for it by a method we call *Search and Confirm*:

In the "search" phase, the user makes a hypothesis about how cases might be described that belong to the focus issue. This hypothesis is specified in a flexible query language that allows the expression of words, strings, regular expressions, and other patterns. Typically, this will be as straightforward as searching for, say, "batt*" to find cases about battery issues. Such queries may be restricted to specific data fields associated with the cases or applied to all default fields.

Note that we do *not* ask the user to define static keywords for a category or, worse, to create a keyword-based rule to describe the category. Nor are the search terms necessarily used as features by the resulting categorizers. They simply provide a way to identify possible training examples.

The cases matching the query are "scooped" out of the full set of cases (both labeled and unlabeled) and a sample of them are displayed. Also on the display is a column of checkboxes for the focus issue, as well as an indication for each of any issues that have already been associated with each case.

The user's task now is to "confirm" that the cases retrieved are actually associated with the issue and to identify any mistakes. This is done by means of the check box, which can have three states: yes, no, and unknown. It should be stressed that unlike the traditional process, in which the user is asked to recognize positive examples out of a long series of mostly negative examples, it is psychologically far simpler to look at a sample of mostly positive examples and pick out the few that don't belong. This task is further simplified by the fact that any matching cases that are already labeled with another issue are inferred as being negative for this issue. This means that the checkbox will have a negative indicator and the user's eye will be drawn to the case. This inference is rebuttable: the user can override it during confirmation. Note that this does *not* remove the positive label for the initial issue: cases can be positive for more than one category in our system, as real-world cases not infrequently stem from more than one problem.

When the user's attention is focused on an exception, it is sometimes obvious what its correct labeling should be. The user can type (unique prefixes of) its correct issue name(s) to label it, creating the issue(s) if necessary. It is then inferred negative for the focus issue, as well as many others. This enhances the training set far better than just labeling it negative for one issue, but this step is completely optional.

As the user looks over the cases being confirmed, typically other descriptive phrases jump out (e.g. 'power' and 'hold a charge' seem to be mentioned a lot) and these can form the basis for subsequent searches.

As soon as the user has labeled cases as being positive for some issue, the system begins retraining classifiers for all affected issue categories and classifying all of the cases. The Naïve Bayes classifiers used are ones that are trivial to retrain and apply, and the process runs in the background idle time, restarting whenever the user changes the training set and goes idle. The user is rarely aware that it is happening, but only notices that the estimated number of cases associated with each issue, displayed along with the issue hierarchy at the top left, updates soon after new cases are labeled or labels are changed.

For fast interactive retraining and efficient leave-one-out cross-validation, we use binomial Naïve Bayes classifiers with Bi-Normal Separation feature selection [8], but for offline use as a high-quality categorizer we use a Support Vector Machine [10] trained using the same training cases. Our study [7] found that Naïve Bayes often performs better in the earliest stages of training a newly created category interactively, when there are very few positives against many negatives inferred from the pre-existing categories.

When determining the training sets to use for the classifiers, inference rules are used to ensure that—lacking other information—cases that are manually labeled as positive for a child category are considered positive for its parent category; cases that are negative for a parent category are negative for its child categories; and cases that are positive for a category are negative for its siblings and cousins. The actual inference logic is complex, but the end result is that it upholds the Principle of Least Astonishment: users are rarely surprised by it.

The system keeps track of manually-assigned (and inferred) labels separately from those determined by the various classifiers, and both sorts of labels can be used in searches. This allows the user to search for "cases classified as e-mail problems", allowing the user to confirm whether they are mostly correct. It also enables searches for "cases classified as screen problems that don't contain the word 'screen' in them" or "cases classified as battery problems that aren't manually labeled as battery problems." Thus, the user can quickly confirm unlabeled cases that the classifier is getting right (and correct the ones where the classifier is wrong), allowing these cases to be used as training examples for further iterations of the classifier. Also useful is the ability to request that the display be sorted based on the likelihood estimate returned by a given classifier. Looking at the list, the user can easily tell the system that the cases at the top of the list are, indeed, (mostly) positive cases for the category and the ones at the bottom are (mostly) negative.

With this interactive iterative technique, it typically does not take long to amass enough positive and negative training examples to be reasonably confident that the resulting classifier has captured the basic concept. The user can then turn to building up a training set for another issue. Should the user come across further instances of the first issue, he or she can easily add them to the training set.

Note that this method does not focus on priors for the categories. The emphasis is on quickly getting a reasonable number of examples for any category and on making it easy for the user to confirm whether examples reflect a focus category.

When training sets have been built for all identified issues, the user then turns back to the task of finding other issues in the data set. This is done by querying the *residual*—those cases that the classifiers in the system cannot confidently assign to any issue. As with looking at the initial list of cases, one or two new issues are likely to be obvious. As with any query, the user can request that the results be clustered and the top words and phrases in the results be displayed, and this can give further guidance about what seems to be in the remaining cases.

As the process progresses, it may become apparent that the hierarchy is not quite correct. The user may, for example, decide that there are a number of issues that could reasonably be considered software issues while another set are hardware problems. On the other hand, the user might decide that a particular issue isn't really a software defect but rather a documentation deficiency. The system makes it straightforward to add new categories, move categories around, and even merge and delete categories without losing work (including work performed by in other sessions and by others working concurrently on unmodified hierarchies), the training examples flowing to where they logically belong, with the work further done in the current session (according to the new hierarchy) being largely useable by users still preferring to use the old hierarchies. The hierarchy is represented as a directed acyclic graph rather than as a tree, so a single issue may be asserted to be an instance of more than one general issue, and links to such general issues may be added and removed, with the system adjusting training sets accordingly.

The system also makes it straightforward to have multiple independent sub-hierarchies, as, for example, one for problem type, another for product line, and a third for customer type. The independence of these hierarchies means that while the system will tentatively infer that a call that has been labeled as an instance of a battery problem is therefore probably not a keyboard problem or a screen problem, it will not similarly infer that the product is not a laptop or the customer is not a small business customer. When it makes sense to do so, sub-hierarchies can be marked as containing mutually exclusive categories, and inference will change to guarantee that exactly one (alternately at most one) issue is chosen for a given case within that sub-hierarchy.

One common hierarchy-modification task occurs when the user notes that there are so many cases associated with a given issue (either by manual labeling or automatic classification) that it is likely that there are different ways of being associated with the issue. For instance, at first the user might be concerned with identifying cases as being power problems, but it might turn out that these partition into cases in which the unit refuses to turn on and those in which the battery does not hold a charge for an acceptable length of time. To find new subcategories, the user can set a *baseline set* of just those cases predicted to belong in the category of interest, and use the previously described techniques—using clusters, frequent words and phrases, and human-noticed hypotheses—to identify subcategories. When focusing on a particular category, the residual cases consist of those that are classified as being in the category but not in any of its children.

This process is used in different ways by different users. Some groups use it to train the classifiers and quantifiers in the interactive tool to generate reports based on the resulting quantifications, either of the number of cases associated with each category or of a quantified estimate of some other quantity (e.g., time or money spent handling calls) associated with the cases. Others use the continually-updating internal classifiers solely as search aids and use the interactive tool as a straightforward way to quickly label manually *all* cases in a data set containing thousands or hundreds of thousands of cases, generating reports for the resulting numbers when they are done. Still other groups use the tool solely as a means of building up training sets for other classifiers used off-line as part of a production process. These other classifiers can be implemented using higher quality induction algorithms that take much longer to train than is workable in an interactive system in which retraining is not an explicit step.

3.3 Quantification

Given a new batch of cases, e.g. monthly or daily, we need to *quantify* how many belong in each category. We only need an accurate count, not the individual classifications. If the classifier is perfectly accurate, then its observed count of positive predictions is accurate. Otherwise, this straightforward method returns poor estimates, biased strongly by the class distribution in training. This problem is concealed by machine learning research practices such as random sampling and cross-validation, which never expose classifiers to shifts in the testing class distribution. Partly as a result of this, research in *quantification* methods is under-explored.

In our business application, changes in the class priors are foundational. Hence, we developed technology to generate highly accurate estimates despite inaccurate classifiers. This is not only necessary—given that perfect text classification is often unattainable with any amount of training effort—but it also helps to further reduce the effort required to generate many analyses. For intuition behind this technology, an insurance company can accurately predict how many car accidents will happen next year, but they cannot predict which cars will have the accidents.

We promote two successful quantification methods: The first simply adjusts the observed count of positives given the *true positive rate* and *false positive rate* of the classifier at thresholds selected for good characterization, rather than accurate classification of individual cases. The other method discards the classification threshold, computes the distribution of scores generated by the classifier on the target set, and fits this curve to a mixture model of the score distributions of positives and negatives determined in training. This latter method can be surprisingly accurate, even in situations with very few positives, e.g. ten. For details, we must refer readers to other papers [5][6].

Beyond simply counting, quantification also involves totaling the hours spent or cost involved resolving each issue type. This can lead to quite a different picture of the top issues, e.g. if a frequent issue is quick to resolve, or a less common issue is very expensive. Again, the straightforward method of classifying cases and then summing the cost of the positives can lead to unacceptable estimates. For this too, advanced quantification methods are called for [5].

Despite our ability to quantify more accurately than individual classifications allow, we find that business users regularly give push-back—they would much prefer to know all the cars that will have accidents. Among the many reasons claimed, one legitimate and persistent demand is to feed the data into a variety of downstream analyses that currently require categorized cases, not just count estimates. For some types of these analyses, it is sufficient to provide a smaller, representative sample of cases on which the classifier has greater confidence, and our system supports this.

3.4 End-to-End Solution

The system was designed as a core engine surrounded by multiple replaceable modules. The engine keeps track of the category hierarchy, case data, and labels and trains classifiers, while the modules allow the system to be customized by plugging in different graphical user interfaces, methods of obtaining and managing data, and report renderers.

For the most commonly used *environment* module, data sets are stored as files on a disk in any of several formats (new formats requiring only a simple adapter to be written) from "single-string-per-line" text files to comma-separated-value spreadsheet files to (for users for whom cases are long documents) ZIP archives. The environment maintains a set of (possibly nested) *projects*, each of which has an associated set of datasets, one or more category hierarchies being developed, and associated with each category hierarchy, training sets reflecting positive and negative labels applied by users in prior sessions. Other environment modules may have other paradigms, for example storing cases in databases and extracting them based on queries.

When a user first brings up the system, they are asked to select a project, then one or more datasets, and finally the training sets that they wish to load. Because the system uses globally-unique identifiers (GUIDs) for categories, it is possible to load training sets that were created while looking at other versions of the hierarchy, with the engine mapping labels that refer to categories in the other version to those in the current version to the extent possible. Because the trainings set explicitly refer to the datasets for the cases they contain labels for, it is possible to have the classifiers trained on cases that are not in the datasets the user specified. The system ensures that such cases are not considered during quantification, although may be displayed as the results of searches, and users may modify their labelings.

The user gets to specify which data fields associated with the cases are to be (1) displayed, (2) used in queries, (3) used by analysis tools such as clustering and frequent word analysis, and (4) used as a source of features by classifiers. They also get to specify the data they wish to see displayed for each category in the hierarchy. Besides the number of positive and negative training cases, the system can display estimates for the total number of cases for the category in the loaded datasets. These estimates can be simple counts based on classification or they can be adjusted by one of the methods described in Section 3.3 and/or corrected to make sure that cases in the training set are counted correctly. Also available is a wide range of statistical measures, from accuracy, precision, recall, and false positive rate to f-measure, information gain, and bi-normal separation.

Once the cases are loaded, the system automatically classifies each case into zero or more of the categories, if any, specified for the project in prior sessions and estimates the count of cases in the loaded data sets for each of these categories. At this point, the user might decide to spend some time refining the category hierarchy or training sets—perhaps after doing a search to identify residual cases (those not confidently classified). Over time, new issues will arise that are not covered by current categories and new ways of describing old issues will appear, and so old categorizers will need their training sets expanded to be able to accurately categorize them.

When the user is satisfied that the categories are reasonable and the categorizers well-trained, the next step is to generate a report that summarizes the current data set. While the system allows arbitrary report generators to be added, the most common way reports are generated is by having the system save an XML file that contains the category hierarchy along with the estimated number of calls for each category and a sample of examples. This XML file is then rendered as an interactive web page, allowing users to view trend graphs of issues over time to see which issues are growing or shrinking in importance, hierarchical bar graphs of category quantifications ordered to identify the most pressing issues, and, when a portion of a hierarchy is sufficiently mutually-exclusive for this to be reasonable, pie charts to quickly show the portion of the calls that are due to various issues.

When an issue has been identified as sufficiently important to warrant allocating engineers or technical writers to address, the extracted sample of example cases for each category (chosen based on both strength of confidence in classification and dissimilarity from one another) are useful for them to get a feel for what the real issue is.

4. DEPLOYMENT & RESULTS

Our system, known as "Incident Categorization & Analysis" (ICA), has been deployed by the HP Technology Solutions Group Worldwide for internal use throughout HP by different product groups. A team in the Knowledge Management section provides training, help desk functions, web-hosting of reports, development and ongoing software support. There is ongoing work with several business units to integrate ICA into their processes, potentially eliminating the need for their call agents to manually assign issue codes to every case.

ICA is also being used to build up classifiers for sorting technical documents into topic areas, e.g. to aid with content migration when consolidating knowledge bases. The choice of ICA was mainly due to its user interface making it easy to build up training sets and train classifiers. In particular, with a unified view it handles multiple, orthogonal classification hierarchies to be applied independently and simultaneously.

While we cannot fully share the impact ICA has had for HP, we can give a few anecdotes from the analysis of the iPAQ product line. ICA identified a major source of call volume that had been completely missed by previous analyses of coded records, partly because the issue did not require repair of the product. Once this issue was identified by our toolset, documentation was generated for customer support, and diagnostics were coded into HP Instant Support troubleshooting agents. This led to a documented return on investment in the following months—the quantitative measurement of which was also enabled by ICA.

We have also seen the converse, issues that were believed to be prevalent but that really were not. For example, there had been a hunch among managers close to the iPAQ product that wireless network connectivity was a common issue. By using ICA, in less than an hour an analyst was able to determine that customers rarely called about this issue, which helped to direct resources to the most urgent issues. To obtain such information via manually assigned codes would have been very slow and expensive: a new *wireless* category would need to be added to the issue-code choices, and agents worldwide informed about the new category; the first month's valid sample would be available for analysis a month after deployment, which could take several months. As one manager observed: "I like it – it helps me win arguments."

5. DISCUSSION

There is nothing like real-world usage of your tool set by eager but inexperienced users to give us data mining researchers a healthy perspective. Here we share some of the open issues and lessons learned.

ICA is being applied to other types of data as well, including query logs, web support forums, and customer comments. Interestingly, it becomes very difficult to tease apart the categories for customer comment email. This is partly because such email can touch upon many different categories, and partly because email messages about a single category are each written by a different user with his or her unique turn of phrase. This dispersion in the feature space makes clustering harder and also requires more training for a classifier. By contrast, call agents tend to re-use phrases for repeat issues they have entered in the past. Since agents communicate with each other, they tend to develop communal terminology and abbreviations.

Even so, when selecting a monthly batch of cases for a given product line, the text features are naturally dispersed because we have call centers that work in English, Portuguese, Korean, Japanese, and Chinese, among others. While the underlying programming language, Java, can easily handle search terms and even regular expressions in these languages, this nonetheless increases the disjunctive training needs to search for and identify training cases in each language for every class. Some languages, particularly Chinese (in which words are not separated by spaces) posed special problems, and the system needed to be augmented with special techniques (some of which we developed) for extracting words to use as features. Our architecture supports having several analysts simultaneously label cases in different languages for a single category hierarchy, rather than depend on cross-lingual information retrieval, which has fairly poor performance even in state-of-the-art techniques [e.g. 15]. This architecture is also needed for large category hierarchies whose diversity of topics requires several people with different domain expertise to label training cases simultaneously.

5.1 Open Issues

What people often want is something for nothing, i.e. very little human effort to generate a decent but imperfect analysis. Our original transfer partner even thought the clustering portion alone would be sufficient for their needs. Unfortunately, neither word counting nor the variety of clustering techniques we tried prove satisfactory for many technical texts, particularly for product lines with long texts describing extensive diagnosis and repair. The failure is partly due to feature dispersion mentioned above, but also due to poorly separable data—many cases involve the same types of troubleshooting steps and interaction with the customer. This is a potentially valuable area for ongoing research, but is unfortunately hindered by the subjectivity of the goal and the widespread focus on clustering clean texts, such as news articles, rather than ‘dirty’ technical text.

There is a trade-off about local versus remote computation. For this highly interactive application, one demands both the quick response of an editor running locally as well as the vast computing and storage resources of servers in a data center. We designed our solution to run on the user’s workstation, and tolerate some startup delay given the long interactive session that follows, but we find there are many users who wish to run it on their laptops against datasets that grow monthly. But performing the memory- and compute-intensive data mining at a server leads to unwelcome network delay, especially if implemented via modern web technologies.

Although the process to build up the category hierarchy and training examples is conceptually simple, in practice it is quite hard for novice users, especially the majority who are not data mining specialists. There are several reasons for this. The simplest is just being unfamiliar with the menus and features, as with any new software. But it is unlike just learning a new editor, where you know the concepts (e.g. boldfacing) but need to learn the commands for them. Instead, it is a first-of-a-kind application. Our users have to learn the concepts of clustering, classification by machine learning, and quantification. They initially don’t know which of the many fields of their datasets they should analyze or allow the classifier to predict from. For example, one field gives the issue code manually assigned to the case by the call agent, but if this field is to be eliminated in the future, the classifier should not depend on it as an input feature.

Moreover, even for a machine learning expert, it can be hard to know just what to do. For example, to improve recall, an expert might know to label additional positive training examples. One solution may be to implement active learning methods [e.g. 11,14], but such methods systematically present the user with some of the hardest to label cases, probably worsening the cognitive load issue. Further, active learning only considers the choice of which item to label next, not the broader problem of which other actions might be more beneficial: For the example of trying to improve recall, we find the problem often lies instead in mistakes in the training set. If some positive items are mistakenly labeled as negative, then the classifier learns to be very conservative in voting positive. Also, if the classifier predicts correctly for some negative items that are wrongly labeled positive, the measured recall will be artificially low.

Finally, even for an expert with a deep understanding of how to most effectively build up the training sets, two serious issues remain. It is hard to optimize the use of one’s time: there are several different ways to improve any given classifier, multiplied by 25–50 categories to develop. One can get stuck in a rat-hole finding interesting sub-clusters for a category or working on catching variant misspellings/language used, when there are much bigger issues elsewhere. There may be a large undiscovered category that would account for a larger fraction of the dataset, or there may be a smaller undiscovered category for which the business would be better able to take action to resolve.

5.2 Lessons Learned

Our initial vision was to develop a tool set that would enable the many domain experts to perform independent analyses of each product line. Beyond the need for having trainers to teach them the tool and to be available for ongoing questions, experience has taught us that there is an essential need for expert knowledge architects. This person should ideally have skills in information architecture and data mining. Their role is to collaborate with the domain expert to establish which fields to analyze and to develop a usefully structured category hierarchy. The domain expert can then work more independently, labeling more training cases and defining additional categories.

This need, as well as the complexity of the task and the limited time available by domain experts for data analysis, has convinced us that just developing a tool set cannot succeed. It needs to be couched within a service offering support and/or full service.

We have been pushing classification technology for years and have never found the user pull and word-of-mouth excitement we have gotten from this work in just quantifying how many items belong in each category. It is a sort of ‘killer app’ for business.

We had thought that trending over time would be highly valued, especially as it requires minimal human effort compared with the initial analysis. In practice, most users already get a lot of value just from an initial quantification analysis. In fact, some users value our software even ignoring its automated quantification capability. For example, when reading a small volume of cases, e.g. fewer than 500 customer survey emails per month for some product line, it proves useful just to keep track of the category hierarchy and the hierarchical tally of issues, even if entirely manually assigned. Moreover, some users were excited just to have a good graphical user interface for viewing cases and performing various searches on them, ignoring the machine learning technologies altogether. As Ron Kohavi stated at his ECML'05 keynote, it is surprising how delighted many clients are with simple first order statistics. This can be astonishing to data mining researchers, who seek to develop sophisticated predictive models.

6. CONCLUSION & FUTURE WORK

The core [mining] algorithms are now a small part of the overall application, being perhaps 10% of a larger part, which itself is only 10% of the whole. –K. Thearling [17]

Supervised machine learning for text classification has been around for decades. Even so, its adoption for real-world tasks is severely limited, partly by its heavy requirement for labeled training cases. For applications like ours where thousands of classifiers need to be trained, the demand for training data was nearly prohibitive. This in turn led us to develop an essential trio of methods: clustering for issue discovery, interactive search-and-confirm training with category inference rules, and quantification. One of the main contributions of this work is in greatly lowering the cost of building up training sets. Before this, many potential classification projects we considered internally were starved by a lack of training data. Now we can examine many more problems offered to us that have only unlabeled datasets.

Considering future research directions, *concept drift* is the bane for classification, and in an application like ours, it is institutionalized, e.g. with ever changing support issues, product models, and their technical environment in deployment. Also, we have recognized *quantification* as an important topic for machine learning research, and encourage other practitioners to share variant settings that are important for them, which may help direct productive research.

Besides the research path, there is strong interest from HP's customers and partners in this capability. We are exploring integration with HP's product offerings (for example, with HP's management software products) and we are actively collaborating with HP Services on this project.

7. ACKNOWLEDGMENTS

We thank Tom Tripp for the original vision that led to this project and his drive in making it real. Shelen Jain and Umesh Dayal have given us many helpful suggestions and support. Jim Stinger has actively participated in this research, particularly with the clustering work. Many others have contributed to the success of this project, particularly Farzana Wyde, Sue Bills, Ron Plourde, Stephane Chiocchetti, Philippe Schmidt, Chas Goebel, Rafael Reyes, Rob Fritz and Jamie El Fattal.

8. REFERENCES

[1] Banerjee, A., Krumpelman, C., Ghosh, J., Basu, S., and Mooney, R. J. Model-based overlapping clustering. In *Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining* (KDD, Chicago), 532-537, 2005.

[2] Beil, F., Ester, M., and Xu, X. Frequent term-based text clustering. In *Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (KDD):436-42, 2002

[3] Deerwester, S., Dumais, S., Furnas, G, Landauer, T, and Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[4] Fawcett, T. and Flach, P. A response to Webb and Ting's 'On the application of ROC analysis to predict classification performance under varying class distributions.' *Machine Learning*, 58(1):33-38, 2005.

[5] Forman, G. Quantifying trends accurately despite classifier error and class imbalance. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD, Philadelphia), 2006.

[6] Forman, G. Counting positives accurately despite inaccurate classification. In *Proc. of the 16th European Conf. on Machine Learning* (ECML, Porto):564-575, 2005.

[7] Forman, G. and Cohen, I. Learning from little: comparison of classifiers given little training. In *Proc. of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD, Pisa):161-172, 2004.

[8] Forman, G. An extensive empirical study of feature selection metrics for text classification. *J. of Machine Learning Research*, 3(Mar):1289-1305, 2003.

[9] Havre, S., Hetzler, E., Whitney, P., and Nowell, L. ThemeRiver: visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9-20, 2002.

[10] Joachims, T. Text categorization with support vector machines: learning with many relevant features. In *Proc. of the 10th European Conf. on Machine Learning* (ECML, Berlin):137-142, 1998.

[11] Li, X., Wang, L., and Sung, E. Multilabel SVM active learning for image classification. In *Proc. of the Int'l Conf. on Image Processing* (ICIP), 4:2207-2210, 2004.

[12] MacQueen, J.B. Some Methods for classification and Analysis of Multivariate Observations, In *Proc. of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, Univ. of California Press, 1:281-297, 1967.

[13] Mei, Q. and Zhai, C. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining* (KDD, Chicago): 198-207, 2005.

[14] Melville, P. and Mooney, R. Diverse ensembles for active learning. In *Proc. of the 21st Int'l Conf. on Machine Learning* (ICML, Banff), 584-591, 2004.

[15] Rogati, M. and Yang, Y. Resource selection for domain-specific cross-lingual IR. In *Proc. of the 27th Annual Int'l ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR, Sheffield), 154-161, 2004.

[16] Suermondt, J., Kirshenbaum, E., Forman, G., and Stinger, J. The 10-second answer: practical text clustering for topic discovery. Forthcoming. HP Labs, Tech.Rpt. HPL-2006-41.

[17] Thearling, K. Some thoughts on the current state of data mining software applications. Workshop: *Keys to the Commercial Success of Data Mining, 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining* (KDD, New York), 1998.