FeedMe:
A Collaborative Alert Filtering System

Shilad Sen*, Werner Geyer**, Michael Muller**, Marty Moore***,
Beth Brownholtz**, Eric Wilcox**, David R Millen**

*University of Minnesota
200 Union Street SE
Minneapolis, MN 55455
+1-612-377-2127

**IBM T.J Watson Research
One Rogers Street
Cambridge, MA 02116
+1-617-693-4791

***IBM Software Group
Five Technology Park Drive
Westford, MA 01886
+1-978-399-6989

sens@cs.umn.edu, werner.geyer@us.ibm.com, michael_muller@us.ibm.com,
martmoor@us.ibm.com, beth_brownholtz@us.ibm.com, eric_wilcox@us.ibm.com,
david_r_millen@us.ibm.com

ABSTRACT
As the number of alerts generated by collaborative applications grows, users receive more unwanted alerts. FeedMe is a general alert management system based on XML feed protocols such as RSS and ATOM. In addition to traditional rule-based alert filtering, FeedMe uses techniques from machine-learning to infer alert preferences based on user feedback. In this paper, we present and evaluate a new collaborative naïve Bayes filtering algorithm. Using FeedMe, we collected alert ratings from 33 users over 29 days. We used the data to design and verify the accuracy of the filtering algorithm and provide insights into alert prediction.

Categories and Subject Descriptors
H.5.3 [**Group and Organization Interfaces**]: *Collaborative computing, Computer-supported cooperative work, Evaluation/methodology.* G.3 [**Probability and Statistics**]: *Experimental design, Probabilistic algorithms*

General Terms
Algorithms, Experimentation, Human Factors.

Keywords
Alert filtering, collaborative filtering, attention management, interruption management, Bayesian, Activity Explorer

1. INTRODUCTION
In a recent survey of 1000 senior executives, Basex estimated that unnecessary interruptions consume about 28% of knowledge worker's day and cost U.S. companies $588 billion per year [36]. Other studies of interruption-frequency provide similar results (e.g. [8], [10], [22], or [35]). Through direct observation, Mark et al. [22] calculated that 57.1% of task segments ("working spheres," in their words) were interrupted. In addition, 22.8% of tasks were *not* returned to within the same work day. Speier et al. [35] showed that interruptions degrade performance on more complex tasks.

Taken together, these studies show that the increased use of technologies in the workplace introduces interruptions that impact productivity. In the past, interruptions were generally limited to phone calls and office coworkers. Desktop computers generate email alerts, instant messaging windows, alert bubbles in the system tray, reminder dialogs, blinking icons, automated system updates and so forth [14]. The likelihood of completing a task without being interrupted is low.

In particular, collaborative desktop applications add to the flood of alerts by notifying users when other people interact with shared artifacts. Although this kind of awareness may be beneficial (it may reduce a user's latency in responding to a request), too many alerts has been shown to be undesirable to users [29]. Interruption management research aims to solve this problem by filtering "good" alerts from "bad" alerts. Determining whether to show an alert, and how and when to show it, depends on many factors such as relative urgency and importance, end user's interest, and user activity and context.

Drawing on early work on semi-structured messages in Information Lens [21], many applications provide simple preference settings or rules to manage alerts. However, setting up explicit rules is often a time-consuming task that needs to be repeated for each individual application. Alternatively, machine learning techniques can potentially be used to automatically filter alerts with relatively little human effort. In this paper, we explore using collaborative filtering, which leverages relationships among users, to differentiate desirable alerts from undesirable alerts. We present a series of naïve Bayes algorithms, including a new collaborative naïve Bayes algorithm that yields 73% filtering accuracy.

In order to test our system's effectiveness, we displayed 16351 alerts to 33 users and asked them to rate the alerts. The users provided a total of 6385 ratings or rating-interpretable responses. The alerts were collected using FeedMe, a novel alert management infrastructure based on alert feeds such as RSS or ATOM. We used Activity Explorer [29] as a test application. In

CSCW'06, November 4–8, 2006, Banff, Alberta, Canada.

Activity Explorer, users collaborate around shared activities. Activities provide a rich context and structure for a managing a task or project.

Our alert filtering task bears similarity to spam filtering in email (e.g., [1], [18]). However, the problem of spam filtering differs from alert filtering in several ways: the alert classification task is more difficult, we explore algorithms that are collaborative in nature, and we explore the use of contextual attributes related to past alerts such as contextual burden. We discuss these differences and other related work on interruption and attention management in Section 2. In Section 3 we describe design decisions and architecture of the FeedMe alert management system. Section 4 presents three filtering approaches, including our collaborative approach. In Section 5 we present results from the data we collected. In Section 6 we discuss limitations and implications of our results.

2. RELATED WORK

A wealth of research is available related to notification systems. In research related to notification system architectures, Rosenblum et al. [31] develop a design framework enumerating seven design dimensions of large-scale notification systems. Cabrera et al. [3] present a distributed notification architecture demonstrating global internet scalability. Carzaniga et al. [6] evaluate a series of architectures for large-scale notification systems, including hierarchical and peer-to-peer designs. The Elvin notification system introduced a distributed subscription system that allowed for fine-grained *quenching* of upstream subscriptions through pattern matching [32]. Although FeedMe supports subscriptions to multiple alert channels, the focus of our work is on filtering alerts rather than efficiently transporting publish / subscribe data.

From an HCI perspective, McFarlane [25] evaluates four strategies of interruption: immediate, negotiated, mediated, and scheduled. McFarlane found negotiated strategies, where the user postponed interruptions for a particular amount of time, to be most effective and well-liked. McFarlane et al. [26] also present broad background and motivation for the field of interruptions research. While McFarlane's efforts evaluate delivery strategies, we focus on the relative user interest level of different alerts.

Of course, not all interruptions are undesirable ([8], [15], [22], [37]). A useful interruption management system should prevent unwanted interruptions while passing through desired interruptions. McCrickard et al. [23] suggest that the notification task can be framed as a cost benefit analysis: a system must determine if the information benefit conveyed in an alert is greater than the cost of interrupting the user. Many researchers have investigated automated techniques for predicting the cost of interruption. Horvitz et al. [15] built Bayesian network models for predicting the cost of interruption based on factors such as the time of day, the current application, and desktop activity. Our work focuses on inferring the benefit side of the equation, an area with far less existing research.

In an approach similar to ours, Horivtz et al. consider methods for classifying the criticality of email messages as a component of their Priorities system [16]. Our work differs in several respects. First, we incorporate direct ratings feedback from users, while the Priorities system is trained on data created by domain experts. Second, whereas the Priorities system is based on a single global classification model, we explore predictive models which analyze relationships among users.

Email spam classification techniques provide a mature approach to similar issues (although other HCI applications have also benefited from machine-learning classification approaches [34]). Cranor et al. [7] give an excellent overview of the spam-filtering task, including techniques such as naïve Bayes classification, support-vector machines, and boosting. Our classification efforts differ from traditional spam detection in several important characteristics. First, the majority of email messages considered spam are unsolicited messages sent by anonymous senders. The majority of alerts today are sent because the user explicitly or implicitly subscribed to an alert. This makes it more difficult to differentiate between a desirable and an undesirable alert. Secondly, we explore collaborative algorithms that explicitly model relationships between users to improve classification. Third, the same alert may be classified as useful, or not useful to the same user under different contextual circumstances. For example, if a user has recently received a very similar alert, they may not be interested in receiving the information again. Fourth, filtering alerts is more time-sensitive, since alerts have a real-time character, i.e. algorithms need to scale appropriately. Fifth, spam filtering generally works by treating the words in the *contents* of the message as a long vector of detectable "features" [18] such as in advertisements, whereas alerts typically have relatively few words of content but (unlike spam) relatively trustworthy metadata.

Several researchers have investigated user interface designs for displaying time-sensitive alerts. Bartram et al. [2] evaluate the use of moticons (icons with motion) in communicating peripheral information. McCrickard et al. [24] analyze notification systems with respect to cost of interruption, reaction time, and information comprehension and suggest that awareness interfaces should match the needs of the notification. While we did evaluate user interface designs for alerts when building our system, this paper focuses on predicting the benefit of an alert's informational content.

Several studies have been conducted using Activity Explorer, our test application (e.g. [12], [27], [29]). Muller et al. [29] studied user behavior in Activity Explorer and discovered that users can be overwhelmed with alerts in Activity Explorer. In notification research closely related to Activity Explorer, Carroll et al. [5] examined awareness interfaces for activity-based systems, finding that maintaining awareness of activities requires transmitting far more contextual information than traditional action awareness.

3. ALERT MANAGEMENT SYSTEM

3.1 Design Considerations

The FeedMe system primarily serves as the platform for testing the collaborative alert filtering described in this paper. However, the design of the system was also driven by the goal of providing a generalized alert management solution that integrates alerts from various sources including desktop applications such as Activity Explorer [29]. We built a centralized system with an *alert management server* that acts as an alert aggregator. Centrally managing alerts has several advantages for us: (1) alert filtering algorithms can be collaborative, i.e. alert feedback from

other users can be taken into account, (2) users can access their alert history from any device, and (3) a user's alert preferences (and filtering) remain the same between different devices unless they are device specific. In order to allow for a flexible integration of alert sources from a wide array of applications, we decided to not only support standard formats such as RSS and ATOM but also provide a way for legacy desktop applications to publish their alerts to an alert *desktop monitor*. The desktop monitor funnels alerts back to the alert management server for filtering purposes, and it also displays alert bubbles on the desktop.

3.2 System Description

Figure 3 shows how the alert management server works in conjunction with the client desktop monitor.

Our server manages a list of alert provider subscriptions for each user. Users can subscribe to new alerts feeds through the web-based management user interface shown in Figure 2. Each alert feed constitutes a "channel" of incoming alerts for a user.

Applications can publish alerts by:

- Providing a public ATOM / RSS URL (e.g. external web applications not shown in the figure),
- Notifying the alert server directly via its Web Services API, or
- Publishing the alert to the desktop monitor who forwards the alert to the alert server.

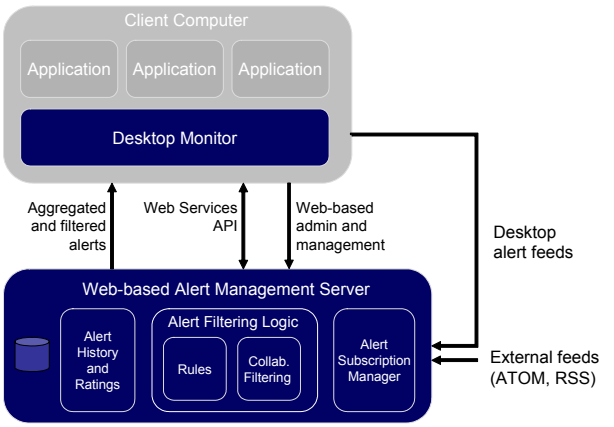

Figure 3. FeedMe System Architecture

Figure 2. FeedMe Alert subscription web UI.

All incoming alert feeds are first filtered based on explicit user-defined rules, (i.e. if a rule applies to a particular alert, the rule fires). If no rule applies, we use collaborative filtering (as described in Section 4). Remaining alerts are forwarded to the desktop monitor who in turn shows the alert to the end user by displaying an alert bubble[1] (see Figure 1).

The alert server also stores end users' ratings of alerts as input for the collaborative filtering algorithm. Users may rate alerts in alert bubbles shown on their desktop by clicking the "thumbs-up" or "thumbs-down" buttons depicted in Figure 1 [28]. A single user action (a click on the buttons) both (a) closes the alert bubble and (b) records the user's rating of the alert. In addition to explicitly rating an alert, we also implicitly create ratings when they select the hyperlink ("Test Activity," in Figure 1) inside the alert navigating to the source of the alert (coded as "thumbs-up"). Clicks on the "x" close box are not interpreted as either "thumbs-up" or "thumbs-down". The desktop monitor stores the alert ratings on the server using the Web Services API.

Figure 1. FeedMe Alert bubble.

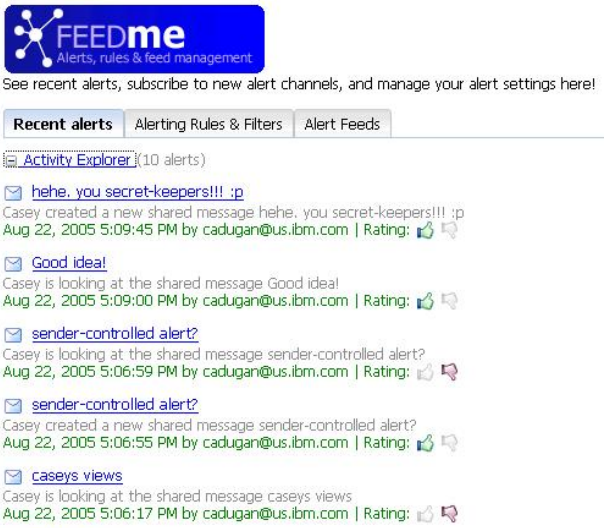

Figure 4. Web UI showing alert history.

[1] This general approach (rules first, statistical filtering second) is similar to some spam-filtration approaches (e.g.,[33]). However, as noted above, our statistical filtering involves historical / contextual information that is not part of conventional spam approaches.

A secondary way of rating alerts is through the web-based user interface of the alert management server (see Figure 4). The server stores a history of all incoming, unfiltered alerts and the web UI can be used to view the history and rate alerts. Since desktop alert bubbles are ephemeral, providing an alert history also allows users to view missed alerts.

The Web UI of FeedMe also allows users to explicitly set up filtering rules such as "Always Show Alerts Containing Some String". Alternatively, applications can use the Web Services API to programmatically create rules on the FeedMe server. Since the focus of this paper is collaborative alert filtering, we do not describe rule management in greater detail here.

3.3 Implementation

FeedMe is implemented using a wide array of technologies. The desktop monitor is implemented in Java. The alert management server runs as a Java web application on an application server (e.g. Websphere [17]). We used XPath as a representation language for explicit rules, Hibernate for persistency, XML-RPC as a web services interface, JSP, JSLT, and Struts for the web UI.

4. FILTERING ALGORITHM

In this section, we present our collaborative alert filtering algorithm. The algorithm builds on and enhances standard and personalized Naïve Bayes filtering. We selected the NB algorithm due to its rich history as a classification method [19]. We begin by describing the alert filtering task and then describe the application of the Naïve Bayes algorithms to our alert filtering task.

4.1 Alert Filtering Task

An alert management system receives a series of alerts. Each alert is directed to a set of target recipient users. The alert filtering task determines whether the alert should be displayed or suppressed to each of the alert's recipients.

A learning system chooses to display or suppress each alert based on that alert's attributes. Alert attributes fall into three classes:

1. *Intrinsic alert attributes* that describe the alert itself, such as the alert's author, and the subject of the alert.
2. *Contextual alert attributes* that relate the current alert to alerts previously received by the user, such as time since last alert and number of alerts in the last minute.
3. *Environmental alert attributes* that relate to a user's current application context, such as the user's current application focus or the time of day [15].

The results in this paper concern the first two classes, namely intrinsic alert attributes and contextual alert attributes. The third category, environmental alert attributes, requires more work to sense the user's current applications and tasks and has previously been explored by Horovitz et. al [15].

Although intrinsic attributes are independent of the recipient user, contextual and environmental attributes are user specific. Thus, the same alert may have different attributes for different users.

Users indicate their satisfaction in receiving an alert by rating the alert. Users may also review and rate suppressed alerts. We chose to focus on binary rating schemes (e.g., thumbs up or down) although systems may select other rating schemes, such as unary and real-valued ratings schemes.

We now formalize the alert filtering task. A system receives alerts $\mathbf{A} = \{a_1,...,a_n\}$ in chronological order. Users $\mathbf{U} = \{u_1,...,u_m\}$ receive alerts through the alert system. An alert a_i is subscribed to by a set of target users: $\text{target}(a_i) = \{u'_1,...,u'_k\} \subseteq \mathbf{U}$. A target user u_i may assign alert a_j a rating value of $r_{i,j} \in \{+1,-1\}$. The set of ratings is represented using the rating matrix $\mathbf{R}$ where rows are users, and columns are alerts. The matrix $\mathbf{R}$ is generally sparse, i.e. most users have rated a small fraction of the alerts. The set of all alert feature attributes (e.g., alert author, alert subject) is denoted by $\mathbf{F}$. The features of a particular alert a_i depend on the recipient user u_j and are denoted by $attr(a_i,u_j) = \{f_1,...,f_z\} \subseteq \mathbf{F}$.

Framed from a machine learning perspective, alert prediction can be viewed as an *online* learning *classification* task. *Online* indicates that our models must dynamically adapt to user feedback in real-time. *Classification* indicates that we are trying to differentiate between two result classes (positive and negative ratings).

4.2 Basic Naïve Bayes Alert Prediction

We begin by decribing the naïve Bayes (NB) prediction algorithm for alerts. We choose to display or suppress a new alert a_i for a target user u_j based on the probability that the user rates the alert a value of +1: $p(r_{i,j}=+1)$. Henceforth we shall abbreviate this probability as $p(r^+_{i,j})$. We condition this probability on the features of the alert:

$$p(r^+_{i,j}) = p(r^+_{i,j} \mid attr(a_i,u_j))$$

Using Bayes rule [13], and an assumption of independence of attributes, we get:

$$\textbf{Eq 1: } p(r^+_{i,j}) = \hat{p}(r^+) \prod_{f_k \in attr(u_i,a_j)} \frac{p(f_k \mid r^+)}{p(f_k)}$$

Where $\hat{p}(r^+)$ is the background prior probability of a positive rating (e.g., the fraction of total ratings that have the value +1), and f_k are alert attributes. Intuitively, Eq 1 multiplies the background probability of a positive rating by the relative likelihood of a positive alert generating the feature, compared to any alert generating the feature.

We similarly calculate the probability of a negative rating:

$$p(r^-_{i,j}) = \hat{p}(r^-) \prod_{f_k \in attr(u_i,a_j)} \frac{p(f_k \mid r^-)}{p(f_k)}$$

We are only interested in relative probabilities for the two classes (+1 and -1). Since the denominator is identical between the two class probabilities, we ignore the denominator. Intuitively, the formula compares the prior probabilities of positive (or negative) ratings multiplied by the fraction of positive (or negative) ratings that contained each of the alert attributes.

We could simply calculate $p(f_k \mid r^+)$, the fraction of positive ratings that contain feature f_k, by measuring the frequency of the feature in positively rated alerts. Unfortunately, the attribute

space is sparse and data from rare attributes can be noisy. We counteract ratings sparsity by treating the probability as a random Bernoulli variable, which naturally models a two-outcome event. We further suppose that the Bernoulli variable parameter is drawn from a Beta distribution, which is the Bernoulli variable's conjugate prior and leads to tractable, closed-form, equations. We incorporate a Bayesian prior estimate of $p(f_k \mid r^+)$ using a beta distribution with parameters α, β. The parameters α, β resemble the relative likelihood of observing or not observing any chosen feature in a positive rating, and can be estimated from the data.

$$\textbf{Eq 2: } p(f_k \mid r^+) = \frac{\left|\left\{r^+ \mid f_k \in attr(u_i, a_p)\right\}\right| + \alpha}{\left|\left\{r^+\right\}\right| + \alpha + \beta}$$

We include a derivation of this equation in the appendix. Qualitatively, this equation smoothes feature probabilities based on the overall average feature probability.

Note that even though real world application attributes do not generally meet naïve Bayes' independence assumption, the naïve Bayes algorithm often performs very well [1].

4.3 Personalized Naïve Bayes

While the basic NB algorithm constructs a single global classifier, the personalized Naïve Bayes (PNB) algorithm constructs a different NB classifier for every user. Unfortunately, moving from a single global model to a model for each user dramatically decreases the number of ratings per attribute. To overcome this sparsity, we alter the beta prior in the simple model in Eq 2 by encoding the global user preference for the attribute into the prior parameters α and β. In essence, PNB smoothes the user-specific probability for a feature towards the overall community probability for the feature.

Our choice of prior depends only on the attribute; not the target user. Given a feature f_k with n^+ positive ratings and n^- negative ratings by all users, we construct the beta prior parameters α_k and β_k as follows:

$$\beta_k = z$$

$$\textbf{Eq 3: } \alpha_k = z \cdot p(f_k \mid r^+)$$

The probability $p(f_k \mid r^+)$ in Eq 3 is calculated directly from the simple naïve Bayes model presented (Eq 2). The parameter z effectively controls the "weight" of the prior distribution. We fix z so the sum of the personal beta prior ($\alpha_K + \beta_K$) is equal to the sum of the global prior ($\alpha + \beta$) estimated in Eq 2. Finally, we calculate the probability of user u_i rating alert a_j positively.

$$\textbf{Eq 4: } p'(f_k \mid r+_{i,j}) = \frac{\left|\left\{r^+_{i,p} \mid f_k \in attr(u_i, a_p)\right\}\right| + p(f_k \mid r^+)(\alpha + \beta)}{\left|\left\{r^+_i\right\}\right| + (\alpha + \beta)}$$

Note that while in Eq 2 the numerator and denominator counted the number of ratings across all users, Eq 4 only counts ratings for the target user u_i. The values of, $p(f_k \mid r^+)$, α and β are all defined using Eq 2. Qualitatively, the personalized model smoothly adapts from the basic model to a more personalized model as the user provides more ratings for an attribute.

We calculate the final predicted rating by inserting the component probabilities of Eq 4 into Eq 1.

Table 1. An alert ratings matrix for four users. (alert attributes are in parentheses)

	Alert 1 (w)	Alert 2 (x,y)	Alert 3 (z)	Alert 4 (z)
Bill	-1	+1	?	?
Jim	-1	+1	-1	?
Martha	-1	+1	-1	?
Fran	+1	-1	+1	?

4.4 Collaborative Naïve Bayes

PNB builds upon NB by incorporating information specific to a user. The collaborative naïve Bayes (CNB) model takes personalization one step further by looking for similarities in attribute ratings across users.

In Table 1 we give a short example demonstrating the usefulness of collaborative algorithms. Assume that Bill receives a new alert (alert 4) with a single attribute *z*. Furthermore, assume that Bill has not rated any alerts with attribute *z*. Jim and Martha, who historically rate similarly to Bill, have rated attribute *z*, and generally rate it negatively. It seems reasonable to infer that Bill will also dislike alert 4, whose sole attribute is *z*. As this example shows, collaborative algorithms can help overcome ratings sparsity.

The CNB algorithm begins by building PNB classifiers for each user as described in section 4.3. Next, for every target user, we construct a higher-level classifier that combines the predictions of every user's personal model. We use support vector machines (SVM) as our second-level classifier [18], and train each user's SVM on the user's history of alert ratings. Input variables for each training instance consist of the output predictions from each user's personal naïve Bayes model. We experimented with a variety of SVM kernels, and selected a polynomial kernel of degree three based on its empirical performance.

Since both a target user's ratings, and other user's predictions change dynamically, we would ideally reconstruct each user's SVM model before every prediction. Unfortunately, this is prohibitively expensive. Instead, we chose to reconstruct a user's collaborative model at exponential rating thresholds (for example, when the user has rated 2, 4, 8, 16, 32, etc. alerts). We found no significant loss in accuracy due to the exponential rebuilding.

5. EXPERIMENTAL RESULTS

5.1 Data Collection

In the summer of 2005, we deployed FeedMe together with Activity Explorer [29] as a test application for alert filtering. Activity Explorer is a collaborative application that supports the notion of shared activities as a way of managing and organizing the context of a project or task. Activities are represented as hierarchically structured collections of shared resources called *activity threads*. Activity threads are created as users collaborate in an activity by posting new shared resources. Users have six different types of activity resources at their disposal to collaborate around and share with other users: message, file, chat, screen snapshot (can be annotated in real-time), folder, and task. A more detailed description of Activity Explorer can be found in [12], [27], and [29]. Activity Explorer provides rich awareness of user

Table 2: Attributes of alerts.

Description of Attribute	Predictive Strength
Intrinsic Attributes of Alerts Themselves	
Resource ID – The unique identifier of the object that generated the alert	0.646
Resource Type – The type of object (chat, document, folder, message, shared screen, task)	0.293
Activity ID – The unique identifier of the activity thread (structured collection of objects) that contained the Resource that generated the alert	0.648
Action – The type of user action applied that generated the alert (view resource, modify resource, add resource, add user)	0.431
Author ID – The person whose action on the Resource generated the alert	0.335
NumberOfMembers – Number of people with access to the Resource that generated the alert	0.173
NamesOfMembers – A list of the names of the people with access to the Resource (decomposed such that each member became a separate predictor)	0.321
Contextual Attributes of the User's Experience Preceding the Alert (with the exception of LastSimilar, all Contextual Attributes are counts of the relevant events, compared with the current alert, in the 60 seconds preceding the current alert)	
LastSimilar – Number of seconds since the most recent alert for the same Resource	0.180
LastMinute – Summary count of number of alerts in the preceding minute	0.105
ResourceConsistency – Count of alerts from the same Resource that generated the current alert	0.138
ResourceDiversity – Count of alerts from different Resources	0.119
ActivityConsistency – Count of alerts from the same Activity whose Resource generated the current alert	0.116
ActivityDiversity – Count of alerts from different Activities	0.119
SocialConsistency – Count of alerts generated by actions of the same Author as the current alert	0.087
SocialDiversity – Count of alerts generated by actions of different Authors	0.144

actions through four types of time-sensitive alerts. Users are alerted when

- New activity resources are created,
- Existing activity resources are modified,
- Users are added to or removed from an activity resource, or
- Users are looking at or editing an activity resource.

Note that these kinds of alerts are only triggered for the members of an activity, i.e. alerts are not public but limited to the member list of an individual activity.

During our study, we disabled all rule-based and predictive filtering in order to collect user feedback about all alerts. This ensured that we could accurately measure the effectiveness of all predictive algorithms during our offline analyses. As a result, the community was flooded with the full spectrum of Activity Explorer alerts (a mean of 481 alerts per user for the participants in this study).

We used the FeedMe Desktop Monitor to tunnel alerts back to the FeedMe server, i.e. when an alert arrives at Activity Explorer, instead of being shown immediately, it is published to the Desktop Monitor, which then sends it to the FeedMe server for processing, from which it is sent to the user. In addition to the standardized RSS attributes, we added custom alert attributes such as activity id, resource id, resource members, and resource type. Note that the FeedMe system has an extension mechanism for alert data sources. This mechanism allows adding custom attributes to the set of attributes that are considered as input for the collaborative filtering algorithm.

5.2 Description of the Data

In order to design and verify the collaborative alert filtering algorithm presented in the previous section, we displayed and collected 16351 alerts during 29 days in a community of 34 users consisting of summer interns, designers, software engineers, and researchers from different parts of IBM. Of these, 6385 alerts received ratings from 33 of the users[2,3] (one user chose not to participate in the rating activity). Overall, the amount of rating activity varied erratically over time, but seemed to track overall application usage. To reduce analysis noise, we removed any

[2] Our observed rate of alert feedback is probably higher than can be expected from real-world applications due to our users' knowledge of, and support for, our data-collection effort.

[3] The other 9966 non-rated alerts fell into several categories. 453 alerts were closed by the user without any other action: We do not know how to interpret the "close" response, so we treated those alerts as non-rated. 9445 alerts timed out and were removed by the system; we do not know whether the user ignored these alerts, or was away from her/his computer. Because we do not know why the user took no action, we treated these alerts as non-rated. An additional 68 alerts received a response to turn off further alerts; to be conservative, we did not interpret this response as a rating. Not included in this analysis were an additional 2558 alerts which could not be displayed to the user because too many alerts were already on the screen. These alerts were never seen by the user, and so of course we did not include these in our analysis.

user who had rated fewer than 20 alerts, or viewed less than 50 alerts. The 20/50 cutoff was chosen to balance the desire to retain ratings against the need to remove users with few ratings that may skew our results. We were left with 6302 alerts from 20 users. We coded the ratings as follows:

Rating = +1, if the user clicked the "thumbs-up" icon *or* the user clicked on the link to the object that had generated the alert (3205 alerts across the 20 users)

Rating = -1, if the user clicked the "thumbs-down" icon (3097 alerts across the 20 users)

Our analyses focused on these rated alerts; however, we used the 9966 unrated alerts to generate the user-specific alert attributes in the categories of *contextual alert attributes*, as noted above.

5.3 Capturing and Calculating the Attributes of the Alerts

Each alert was characterized by a set of attributes. Table 2 presents the definition of those attributes, in the two broad categories of intrinsic attributes and contextual attributes.

The **intrinsic attributes** were defined at the moment of *creation* of the alert, and were the same for all recipients of the alert. These attributes included, for example, the resource that generated the alert, the activity thread (structured collection of resources) that contained the resource, the user whose action generated the alert, the type of action that the user took, and so on. See Table 2 for a complete list.

The **contextual attributes** were defined at the moment of *receipt* of the alert, and were different for each recipient of the alert. These attributes included aspects of the user's experience during the 60 seconds preceding the display of the alert, such as the number of alerts with one or more attributes that were the *same* as the current alert ("consistency" attributes) and the number of alerts with one or more attributes that were *different* from one another ("diversity" attributes). For example, ResourceConsistency was a measure of the number of alerts in the preceding 60 seconds that were generated from the *same* resource as the current alert. By contrast, ResourceDiversity was a measure of the number of alerts in the preceding 60 seconds that were generated from *different* resources. See Table 2 for a complete list.

5.4 Filtering Accuracy

We evaluated the performance of the three alert filtering algorithms using the ratings collected with FeedMe and Activity Explorer as a test application (see Figures 1 - 4). All filtering algorithms used the attributes described in Table 2.

We begin our accuracy evaluation by separating our data into *test* and *training* sets. Since we performed ten-fold cross validation, each test set contained 1/10 of the data and the training set contained the remaining 9/10.

Next, we step through all alerts. If the alert belongs to the test set, we predict the rating value using the predictive model and store the result. We then incorporate the alert into the predictive model, regardless of whether it belongs to the test or training set[4]. Note that we took caution to make the accuracy evaluation as realistic as possible. In particular, when predicting the classification of a target alert, we only build the model using alerts that occurred before the target alert.

Based on the results of a seven-fold cross validation repeated ten times, we create confusion matrices comparing the counts of actual and predicted positive and negative ratings (see Tables 3 – 5). These matrices allow us to assess the overall accuracy of each method of predicting users' ratings. As an example, the upper left corner of Table 3 indicates that 28.3% of all alerts were predicted as useful and rated as useful (true positives), while 22.7% of all alerts were predicted as not useful but rated as useful (false negatives). The overall accuracy is the sum of the percentages in cells along the Northwest diagonal.

Simple Naïve Bayes Classifier: The simplest of the filtering approaches, using the same set of predictors for *all* users' ratings, has an overall predictive accuracy of 64% (Table 3, $\chi^2(1) = 531$, $p<.0000001$). This result is similar to results achieved with simple spam filters (e.g. [1], [20]). However, we note that our prediction is based only on alert meta data and the user's recent alert history, 15 predictors in total. By contrast, spam filters are based primarily on the content of each email, which are modeled as a potentially infinite vector of word features as predictors. Our finding of 64% accuracy indicates that simple, scalable, alert classification algorithms can do significantly better than a random baseline.

Personalized Naïve Bayes Classifier: When we enhance the NB model with the ability to personalize each user's predictive model, the overall accuracy increases to 71.8%. (Table 4, $\chi^2(1) = 1225$, $p<.0000001$). The difference between the simple classifier and the personalized classifier is significant, based on a comparison of the correct ratings of each solution (i.e. the major diagonals of Tables

Table 3: Confusion matrix for simple Naïve-Bayes classifier.

Overall accuracy: 64.0%		Predicted Ratings	
		+1	-1
Actual Ratings	+1	28.3%	22.7%
	-1	13.35%	35.67%

Table 4: Confusion matrix for personal Naïve-Bayes classifier.

Overall accuracy: 71.8%		Predicted Ratings	
		+1	-1
Actual Ratings	+1	38.2%	12.7%
	-1	15.4%	33.6%

[4] Note that the predictive process occurs over time, and the prediction of the rating of each alert can only be based on the alerts that have preceded it. Therefore, an alert in the test set should be considered to be part of the history preceding other alerts. Therefore, we do not include a test-set alert in the creation of the model itself, but we do include a test-set alert in the stream of alerts that is used to calculate the history of the ensuing alert.

Table 5: Confusion matrix for collaborative Naïve-Bayes classifier.

Overall accuracy: 73.4%		Predicted Ratings	
		+1	-1
Actual Ratings	+1	38.3%	12.6%
	-1	14.0%	35.1%

3 and 4, $\chi^2(1) = 69.05$, $p<.001$). The additional work of calculating an individual model for each user appears to provide significantly better prediction.

Collaborative Naïve Bayes Classifier: When we combine the personalized models of Table 4 with a weighting of other users' models, the overall accuracy increases a bit further to 73.4% (Table 5, $\chi^2(1) = 1379$, $p<.0000001$). Thus, at a first level of interpretation, we conclude (a) that our modeling can successfully predict users' preferences about alerts, and (b) that an interruption management system with a machine-learning algorithm based on any of these models should help users to reduce the likelihood of undesirable alerts while maintaining likelihood of desirable alerts. We discuss further implications of our accuracy results in Section 6.

5.5 Predictive Strength of Alert Attributes

We now analyze the contributions of particular alert features to overall predictive accuracy. As a point of principle, naïve Bayesian analysis typically does not provide the kind of summary statistical significance that is expected in most social sciences papers: "Bayesian hypothesis testing is often less formal than the non-Bayesian variation. By far, the most common procedure for summarizing results in social sciences research is to simply describe the posterior distribution rather than to apply a rigid decision process." [13] (see also [30]). Indeed, most predictive modeling research in this tradition (e.g., [1], [4], [7], [11], [18]) is directed at the pragmatic task of simply predicting outcomes, and not at the theoretical task of understanding how those outcomes came to be. As a result, these studies usually report a statement of omnibus predictive accuracy (as we did above, in Section 5.4), without interpretation of which components provided the predictive power.

We wanted to provide greater interpretive analysis than the conventional Bayesian approaches. In order to get a rough estimate of the relative predictive strength of different attributes, we calculated the mean user rating for alerts containing each of the attribute values (for example, resource type = 'chat'). We then averaged the absolute value of the mean ratings across the field's values ('chat', 'document', 'folder', etc). Intuitively, if objects have large predictive strengths, values for the field generally have mean ratings that are far from zero. Table 2 includes the mean ratings as an index of predictive strength for all attribute fields.

In brief, the intrinsic alert attributes (Table 2) provided greater predictive strength than the contextual alert attributes (significant at $p<.002$ by the Mann-Whitney test). For our data, it appears that the users' experience preceding an alert is relatively unimportant to the desirability of the alert: Users' ratings appear to depend primarily on the alerts themselves, and not on users' recent history. If this finding is corroborated in other studies, then (a) we can construct a model of user preference that is primarily data-driven rather than experience-driven; and (b) we can focus interruption-management systems on data objects rather than on monitoring users' recent activities. These results can help us evaluate the interruption-management strategies proposed or implied in, e.g., [8], [9], [15], [22], [23], [24], or [25].

Because the distribution of the predictive strength measure is unknown, we are not yet in a position to make statistical comparisons of individual predictive strengths. It appears that the strongest predictors are the collaborative objects themselves (strength = .646) and/or the structured collections in which those objects occur (strength = .648), followed by the type of user action that triggered the alert (strength = .431). Interestingly, this data supports some earlier hypothesis in Activity Explorer [29] that actions such creating or adding member to resources are more important event types than modifying or viewing resources. It appears that the type of object (e.g., file vs. chat vs. message) is less important, and that the author of the event is also less important. However, until we improve our ability to make direct comparisons among predictive-strength indicators, we cannot make claims of statistical significance for these predictor-by-predictor contrasts. We hope to refine these statistical analyses and their interpretations in a future paper.

6. DISCUSSION

Accuracy rates of 73% are probably not good enough for real-world applications. In particular, our users said that they would be unhappy with missing one out of four truly useful alerts. Even though predictive alert filtering might not provide a stand-alone solution to alert management, it should prove useful when paired with rules-based filtering techniques. As mentioned earlier, we designed FeedMe as a hybrid rule based and predictive alert filtering system. Due to time constraints, we were not able to compare rules based, predictive, and hybrid filtering techniques. We hope to perform this analysis in the future[5].

Although we implemented FeedMe as a general purpose alert management system, we only tested its effectiveness in filtering alerts for Activity Explorer due to time constraints. As future work, a cross-application analysis of alert filtering may give some insight into whether we can derive benefits from cross-domain alert feedback. For example, do ratings from Activity Explorer alerts may help in filtering internet news feeds?

As we mentioned in section 5.4, the collaborative naïve Bayes classifier demonstrated only a small (1.6%) improvement in accuracy over the personalized naïve Bayes classifier. Given that CNB requires a more complex system infrastructure and greater computational effort than PNB, it may not seem that CNB is worth the added effort. However, collaborative methods may prove worthwhile in many real-world applications for several reasons. First, our dataset is relative small compared to most real-world datasets. It is likely that, with 10 or 100 times more data, we could detect more subtle patterns and achieve higher filtering accuracy. Second, more sophisticated machine-learning techniques may lead to increased accuracy and reduced computational complexity. Third, Activity Explorer alerts represent a particularly narrow domain. Users already "share" the

[5] The few users that continued using FeedMe after our survey mostly created inclusionary rules for specific users and resources (e.g. "*Always show me alerts from Marty*").

resource generating the alert, and generally have a high level of interest in the shared objects. Many applications, such as internet news feeds, represent a larger, more diverse set of resources that generate more frequent and varied alerts. Predictive filtering is more likely to prove useful in these complex domains.

7. CONCLUSION

Collaborative filtering has been successfully applied in recommender systems to suggest books, movies, and other products to consumers. In this paper, we have evaluated how this learning-based approach can also be used to reduce the noise caused by unwanted interruptions for knowledge workers. We have further shown that the performance of the simple Bayesian model may be enhanced through both user-specific personalization of the predictions, and through collaborative weighting of the individual predictions. Our approach differs from existing interruption management work in that we predict the usefulness of an alert based on the benefit of the alert content, i.e. we are mostly considering intrinsic alert attributes (plus contextual attributes). With this small set of attributes, we were able to predict the usefulness of an alert with an overall accuracy of up to 73.4%.

We believe that this research represents a first step towards predictive alert filtering. A number of factors may improve overall accuracy such as including environmental attributes, collecting more ratings data. We also believe that ultimately only a combination of rule-based and automated filtering will yield the desired level of end user satisfaction. We hope to explore these issues in future research based on FeedMe. At the same time we are currently conducting a deeper analysis of the alert data to disclose more behavioral aspects of interruption management. We expect that this line of research will provide useful insights that help us further improve our collaborative filtering approach by, for example, selecting the most significant alert predictors.

8. ACKNOWLEDGMENTS

We thank the 33 participants in our study, and the members of the Collaborative User Experience Research group at IBM who contributed their ideas to FeedMe.

9. REFERENCES

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., and Spyropoulos, C.D, "An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages," *Proc. of the 23rd Annual international ACM SIGIR Conf. on Research and Development in Information Retrieval*, Athens, Greece, July 2000, ACM Press, New York, NY, 160-167.

[2] Bartram, L., Ware, C., Calvert, T., "Moving icons, detection and distraction," in: M. Hirose (Ed.), *Human-Computer Interaction – INTERACT 2001 Conference Proceedings*.

[3] Cabrera, L.F., Jones, M.B., Theimer, M., "Herald: Achieving a Global Event Notification Service," *hotos*, p. 0087, Eighth Workshop on Hot Topics in Operating Systems, 2001.

[4] Carreras, X Màrquez, L., "Boosting Trees for Anti-Spam Email Filtering," *Conference on Recent Advances in NLP* (RANLP'01). Tzigov Chark, Bulgaria. 2001.

[5] Carroll J. M., Neale D. C., Isenhour P. L., Rosson M. B. & McCrickard D. S. (2003) Notification and awareness: Synchronizing task-oriented collaborative activity, *International Journal of Human-Computer Studies*, 58 (5), 605-632

[6] Carzaniga, A., David S. Rosenblum, D.S, and Wolf, A.L., "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, 19(3):332--383, August 2001.

[7] Cranor, L. F. and LaMacchia, B. A. 1998. Spam!. *Commun. ACM* 41, 8 (Aug. 1998), 74-83.

[8] Cutrell, E., Czerwinski, M., & Horvitz, E., "Notification, disruption, and memory: Effects of messaging interruptions on memory and performance," *Proc INTERACT 2001*.

[9] Czerwinski, M., Cutrell, E., Horvitz, E., "Instant messaging and interruption: Influence of task type on performance," *Proc OZCHI 2000*.

[10] Czerwinski, M., Horvitz, E., & Wilhite, S., "A diary study of task switching and interruptions," in: *Proc ACM CHI 2004*.

[11] Drucker, H., Wu, D., Vapnik, V.N., "Support vector machines for spam categorization", in *IEEE Trans on Neural Networks, 1999*.

[12] Geyer, W., Vogel, J., Cheng, L., Muller, M, "Supporting Activity-Centric Collaboration through Peer-to-Peer Shared Objects," in: *Proc. ACM Group 2003*, Sanibel Island, FL, USA, November 2003.

[13] Gill, J., *Bayesian Methods: A Social and Behavioral Sciences Approach*, Chapman & Hall/CRC, 2002.

[14] González, V.M., Mark, G., "'Constant constant multitasking craziness': Managing multiple working spheres," in: *Proc. ACM CHI 2004*, ACM Press, 2004.

[15] Horvitz E. & Apacible J. (2003) Learning and reasoning about interruption, in: *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI'03)*. New York: ACM Press, 20-27.

[16] Horvitz, E. Jacobs, A., and Hovel, D. (1999). Attention-Sensitive Alerting in: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 99)* 305-313.

[17] IBM Websphere Software, http://www.ibm.com/software/websphere/ (verified 17 March 2006).

[18] Joachims, T., *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

[19] Lewis, D. D. 1998. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *Proceedings of the 10th European Conference on Machine Learning* (April 21 - 23, 1998). Lecture Notes In Computer Science, vol. 1398. Springer-Verlag, London, 4-15.

[20] Madigan, D., "Statistics and the war on spam," in R. Peck, G. Casella, G.W. Cobb, R. Hoerl, D. Nolan, R. Starbuck, & H. Stern (eds.), *Statistics – A guide to the unknown*. Duxbury Brooks/Cole, 2005, Available at http://www.stat.rutgers.edu/~madigan/PAPERS/sagtu.pdf (verified 13 March 2006).

[21] Malone, T.W., Grant, K.R., Lai, K., Rao, R., & Rosenblitt, D., "Semi-structured messages are surprisingly useful for computer-supported coordination," *ACM TOOIS 5*, pp. 115-131, 1987..

[22] Mark, G., González, V.M., & Harris, J., "No task left behind? Examining the nature of fragmented work," *Proc ACM CHI 2005*.

[23] McCrickard D. S., Catrambone R., Chewar C. M. & Stasko J. T. (2003) Establishing tradeoffs that leverage attention for utility: Empirically evaluating information display in notification systems, *International Journal of Human-Computer Studies*, 58 (5), 547-582
[24] McCrickard D. S. & Chewar C. M. (2003) Attuning notification design to user goals and attention costs, *Communications of ACM*, 46 (3), 67-72
[25] McFarlane D. C. (2002) Comparison of four primary methods for coordinating the interruption of people in human-computer interaction, *Human-Computer Interaction*, 17 (1), 63-139
[26] McFarlane D. C. & Latorella K. A. (2002) The scope and importance of human interruption in human-computer interaction design, *Human-Computer Interaction*, 17 (1), 1-61
[27] Millen, D. R, Muller, M. J., Geyer, W., Wilcox, E., and Brownholtz ,B., "Patterns of Media Use in an Activity-Centric Collaborative Environment," in: *Proc. ACM CHI 2005*, Portland, OR, April 2005.
[28] Muller, M.J. (2003), "Method and apparatus for single selection evaluations in interactive systems," United States Patent and Trademark office application 20030085927.
[29] Muller, M.J., Geyer, W., Brownholtz, B., Wilcox, E., and Millen, D.R., "One-hundred days in an activity-centric collaboration environment based on shared objects," in: *Proc. ACM CHI 2004*.
[30] Nicholls, N. "The insignificance of significance testing," *Bulletin of the American Meteorological Society 82*, pp 981-986 (2001).
[31] Rosenblum, D. S. and Wolf, A. L. 1997. A design framework for Internet-scale event observation and notification. In *Proceedings of the 6th European Conference Held Jointly with the 5th ACM SIGSOFT international Symposium on Foundations of Software Engineering* (Zurich, Switzerland, September 22 - 25, 1997). M. Jazayeri and H. Schauer, Eds. Foundations of Software Engineering. Springer-Verlag New York, New York, NY, 344-360.
[32] Segall, B., and Arnold D., "Elvin has left the building: A publish/subscribe notification service with quenching." In Proceedings AUUG97, pages 243--255, Canberra, Australia, September 1997.
[33] Segal, R., Crawford, J., Kephart, J., Leiba, B., "SpamGuru: An Enterprise Anti-Spam Filtering System," In *Proc. of the First Conference on Email and Anti-Spam*, July, 2004.
[34] Sebe, N., Cohen, I., Cozman, F.G., Gevers, T., & Huang, T.S., "Learning probabilistic classifiers for human-computer interaction applications," *Multimedia Systems 10* (6), pp. 484-498, 2005.
[35] Speier, C., Valacich, J.,S., & Vessey, I., "The effects of task interruption and information presentation on individual decision making," *Proc ICIS'97*.
[36] Spira, J.B and Feintuch, J.B., "The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity," Basex, 2005.
[37] SuwatanaPongched, P., "A more complex model of relevancy in interruptions," available at http:// www.spong.org/~pechluck/HCI/content-of-interruptions.pdf (verified 6/24/04).

APPENDIX

A.1 Derivation of Bayesian Estimate of Naïve Bayes Parameters

In Naïve Bayes models, the probability of an alert a_j having a feature f_k given that the alert was rated positively is traditionally calculated by using the observed frequencies, and adding one:

$$p(f_k \mid r^+) = \frac{\left|\left\{r^+_{i,j} \mid f_k \in attr(u_i, a_j)\right\}\right| + 1}{\left|\left\{r^+\right\}\right|}$$

The attribute space for alerts in applications that we studied is often very sparse, and this ad-hoc heuristic does not scale-down well. Instead, we choose a more principled Bayesian approach to parameter estimation. Although the use of conjugate priors has been commonplace in many fields, it is not often used in naïve Bayes algorithms. Thus, we include a brief derivation.

First, we model the conditional probability as a Bernoulli variable with parameter θ. If we know the actual value of the parameter θ, the conditional probably would simply be θ.

$$p(f_k \mid r^+) = p(Bernoulli(\theta)) = \theta$$

Assume that we don't know θ with certainty, but instead have a probability distribution describing our beliefs about the parameter θ. We can calculate the conditional probability as the expectation of the value of θ:

$$p(f_k \mid r^+) = E(\theta) = \int_\theta \theta p(\theta) d\theta$$

We choose a beta distribution with parameters α, β to model the probability distribution on θ. If $B(\alpha, \beta)$ is the beta function with parameters α, β, the probability of a particular choice $\hat{\theta}$ of θ is:

$$p(\theta = \hat{\theta}) = \frac{\hat{\theta}^{\alpha-1}(1-\hat{\theta})^{\beta-1}}{B(\alpha,\beta)}$$

We can now use Bayes rule to calculate the probability of a particular choice of θ given n^+, the number of positive ratings for attribute f_k, and n^-, the number of negative ratings for f_k.

$$p(\theta \mid n^+, n^-) = \frac{p(n^+, n^- \mid \theta) p(\theta)}{\int_\theta p(n^+, n^- \mid \theta) p(\theta) d\theta}$$

The numerator is the product of

- The likelihood of the observed ratings given the choice of θ (a Bernoulli random variable with parameter θ)
- The prior distribution on θ (a beta distribution).

Simplifying, we get a beta distribution with parameters $(n^+ + \alpha, n^- + \beta)$.

$$p(\theta \mid n^+, n^-) = \frac{\theta^{n^+ + \alpha + 1}(1-\theta)^{n^- + \beta - 1}}{B(n^+ + \alpha, n^- + \beta)}$$

The expectation of the beta distribution is:

$$E(\theta) = p(f_k \mid r_{i,j}) = \frac{\alpha + n^+}{\alpha + n^+ + \beta + n^-}$$

We have shown that a Bayesian estimation of $p(f_k|r_{i,j})$ is simply the observed frequency of the attribute f_k combined with the parameters of the beta prior distribution.