A Hybrid Learning System for Recognizing User Tasks from Desktop Activities and Email Messages

Jianqiang Shen, Lida Li, Thomas G. Dietterich, Jonathan L. Herlocker
1148 Kelley Engineering Center
School of EECS, Oregon State University
Corvallis, OR 97331, U.S.A.
{shenj, lili, tgd, herlock}@eecs.oregonstate.edu

ABSTRACT

The TaskTracer system seeks to help multi-tasking users manage the resources that they create and access while carrying out their work activities. It does this by associating with each user-defined activity the set of files, folders, email messages, contacts, and web pages that the user accesses when performing that activity. The initial TaskTracer system relies on the user to notify the system each time the user changes activities. However, this is burdensome, and users often forget to tell TaskTracer what activity they are working on. This paper introduces TaskPredictor, a machine learning system that attempts to predict the user's current activity. TaskPredictor has two components: one for general desktop activity and another specifically for email. TaskPredictor achieves high prediction precision by combining three techniques: (a) feature selection via mutual information, (b) classification based on a confidence threshold, and (c) a hybrid design in which a Naive Bayes classifier estimates the classification confidence but where the actual classification decision is made by a support vector machine. This paper provides experimental results on data collected from TaskTracer users.

Categories and Subject Descriptors: I.2.1 [Artificial Intelligence]: Applications and Expert Systems – *Office automation*

General Terms: Design, Human Factors, Experimentation

Keywords: Intelligent interfaces, machine learning, naive Bayes, support vector machines

1. INTRODUCTION

Knowledge workers, such as professors, managers, and engineers, perform many different activities in their daily work life. The goal of the TaskTracer system is to provide an intelligent user interface that can help such multi-tasking knowledge workers easily organize and access the resources (files, folders, email messages, contacts, and web pages) that they need to support these activities. TaskTracer is based on two main premises: (a) the behavior of the user at the desktop is a mixture of different activities and (b) each activity is associated with a set of resources relevant to that activity. In short, TaskTracer assumes that "activities" provide a useful abstraction for organizing and accessing resources.

In the basic operation of TaskTracer, the user defines a hierarchy of activities that, for historical reasons, we refer to as "tasks." These are typically on-going activities such as "teach CS534," "prepare NSF proposal B," "make travel arrangements for conference C", and so on. Once the tasks have been defined, the user can then easily indicate to TaskTracer the name of the current task (via a drop-down menu that is attached to the currently-active window). TaskTracer records all folders, files, web pages, and email messages that are accessed and associates them with the current declared task. Then various TaskTracer components provide services to the user based on this information. For example, the TaskExplorer provides an easy way to access all of the items associated with a task. The FolderPredictor modifies the default File Open/Save dialogue box so that it is opened in folders most strongly associated with the current task. Users report that TaskExplorer and FolderPredictor are extremely helpful.

A drawback of the current TaskTracer system is that the user must remember to change the current declared task each time the task changes. If the user forgets to do this, then resources become associated with incorrect tasks, and TaskTracer becomes less useful. For this reason, we would like to supplement the user's manual task declaration with a TaskPredictor that attempts to predict the current task of the user. If the TaskPredictor is sufficiently accurate, its predictions could be applied to associate resources with tasks, to propose correct folders for files and email, and to remind the user to update the current declared task.

A variety of recent work has demonstrated the success of machine learning approaches for recognizing human activities [7, 8]. Some commercial applications have been developed [9]. The primary challenge of activity recognition is that the data are quite noisy. There may be irrelevant actions intermixed with relevant ones, the user may do the same task in many different ways, and different activities may involve the same set of objects. For all of these reasons, it is useful to view activity recognition as a probabilistic prediction problem.

In this paper, we describe two machine learning systems for predicting the current task of the user. The first system is TaskPredictor.WDS, and it predicts the current task

IUI'06, January 26–February 1, 2006, Sydney, Australia.

Figure 1: The screenshot of TitleBar, an application that is attached to the window in focus and indicates the current task.

based on properties of the window currently in focus. The second system is TaskPredictor.email, and it predicts the current task based on properties of incoming email messages (sender, recipients, subject, etc.). TaskPredictor.email is particularly important, because each new incoming email message typically results in a task switch, and it is absurd to expect the user to tell TaskTracer the current task for each new email.

The two learning systems are evaluated using data collected from daily use of TaskTracer by 9 users over several months. The results show that TaskPredictor.WDS can achieve more than 80% precision with 10–20% coverage (i.e., proportion of the time that a prediction is made). TaskPredictor.email can achieve more than 90% precision with 65% coverage. These performance levels are quite good, particularly considering the amount of noise in the data resulting from the failure of the user to always remember to update the current declared task.

The remainder of this paper is structured as follows. First, we describe the TaskTracer data collection architectures. Then we describe the machine learning methods that we have applied to the activity prediction problem. Third, we give experimental results from a deployment of TaskTracer within our research group. We conclude the paper with a review of related work and a discussion of the results.

2. ONLINE DATA COLLECTION

TaskTracer operates in the Microsoft Windows environment and collects a wide range of events describing the user's computer-visible behavior. The TaskTracer system is described in detail in Dragnov et al. [6]. In order to collect events with labeled tasks, we devised a special drop-down box in the title bar. A screenshot of this titlebar is shown in Figure 1. This drop-down box shows which task the user is performing and it is attached to the title bar of the window in focus. The user can switch between tasks by selecting a task name from the drop-down box.

TaskTracer collects events from MS Office 2003, MS Visual .NET, Internet Explorer and the Windows XP operating system. An event message contains the following information: 1)*Event type* — such as window focus, file open, file save, web page navigation, and so on; 2)*Listener ID* — the source of the EventMessage (MS Office, file system hook, Internet Explorer, Windows explorer, etc.); 3)*Body* — the detailed information about the event e.g., pathname, window title, email address, Uniform Resource Locator (URL); 4)*Time* — time the event occurred.

From the raw event stream, the main TaskPredictor extracts a sequence of Window-Document Segments (WDSs). A WDS consists of a maximal contiguous segment of time in which a particular window has focus and the name of the document in that window does not change.

In our approach, a new WDS is defined to begin when one of the following events happens:

- *Navigate* (Internet Explorer): the browser displays a new webpage;
- *OsWindowFocus* (all applications): a different window gains the focus;
- *Open* (MS Office): the user opens a new file
- *SaveAs* (MS Office): the user saves a file under a new name;
- *New* (MS Office): the user creates a new blank document.

TaskPredictor.WDS attempts to make a prediction for each WDS. To do this, it extracts the following information from each WDS: the window title, the file pathname, and (for web pages) the website URL. It heuristically segments these into a set of "words" and then creates a binary variable x_j in the feature set for each unique word. If this word appears in the event, x_j is 1, otherwise x_j is 0.

TaskPredictor.email does not use the WDS event stream. Instead, it attempts to make a prediction for each incoming email message. It creates a boolean feature for each observed email sender (the "FROM" field), one boolean feature for each observed *set* of email recipients (the union of the "FROM", "TO", "CC", and "BCC" fields), and one boolean feature for each distinct word observed in the "SUBJECT" field. Note that each set of recipients is treated as a separate feature, so an email message sent to $\{A, B\}$ might have no (true) features in common with an email message sent to $\{A, B, C\}$ unless they were from the same person or had the same words in the subject. We did not find that the words in the email body had any additional predictive value.

3. MACHINE LEARNING METHODS

This section describes the three machine learning methods that we have employed to build our two task predictors: (a) classification thresholds, (b) mutual information feature selection, and (c) a hybrid Naive Bayes/Support Vector Machine classifier.

3.1 Classification Thresholds

In user interface applications, it is essential to avoid annoying the user by making incorrect predictions. Indeed, it is better to make no prediction at all than to make an incorrect prediction. Therefore, we make predictions based on a probabilistic prediction threshold θ. Let $\mathbf{x}$ be the vector of features extracted from the WDS or from the email message, and let y be the task to be predicted. We employ probabilistic learning algorithms that predict $P(y|\mathbf{x})$ and $P(y)$ for each possible task y.

TaskPredictor.WDS uses this information to compute $P(\mathbf{x})$ according to the formula

$$P(\mathbf{x}) = \sum_y P(\mathbf{x}|y)P(y).$$

It then compares $P(\mathbf{x})$ to a threshold θ, and if $P(\mathbf{x}) > \theta$, it makes a prediction. Otherwise, it does nothing. In effect, TaskPredictor.WDS is estimating the probability density of data points in the neighborhood of $\mathbf{x}$. If it has previously observed many data points near $\mathbf{x}$, then it is reasonable to make a prediction, because the prediction is based on sufficient data. If not, it is better to make no prediction.

TaskPredictor.email employs a slightly different method. Let $\hat{y} = \text{argmax}_y P(y|\mathbf{x})$ be the class with the highest predicted probability, and let $\hat{p} = P(\hat{y}|\mathbf{x})$ be its predicted probability. If $\hat{p} > \theta$, then TaskPredictor.email makes a prediction; otherwise, it does nothing. This approach has a different theoretical foundation based on the expected cost of an error. If the cost of a prediction error is α, the cost of a correct prediction is zero, and the cost of making no prediction at all is β, then the prediction rule that minimizes the expected cost is

$$\begin{array}{ll} \text{predict } \hat{y} & \text{if } \hat{p} > 1 - \frac{\beta}{\alpha} \\ \text{make no prediction} & \text{otherwise} \end{array}$$

Hence, $\theta = 1 - \frac{\beta}{\alpha}$.

We will employ two measures of prediction performance: coverage and precision. Coverage is the fraction of cases in which a prediction was made (i.e., $\hat{p} > \theta$). Precision is the fraction of those predictions that were correct (i.e., $\hat{y} = y$, the correct task). For TaskPredictor.WDS, we need high precision but we do not need high coverage. This is because each episode (i.e., each period of time during which the user is working on a single task) is composed of very many WDSs. There is no need to make a prediction for every single WDS. The important thing is that if the user has forgotten to update the task, TaskPredictor.WDS should catch this promptly and make a correct prediction for at least one WDS in the episode.

Similarly, we envision applying TaskPredictor.email to predict the email folder into which the user will want to save the email message. If we make no prediction at all, the user will just experience the standard email foldering dialogue. If we make an accurate prediction, then we can make the email foldering dialogue more efficient. So again, the goal is high precision even at the cost of low coverage.

3.2 Feature Selection via Mutual Information

It is well known that feature selection can improve the accuracy of classifiers by reducing the complexity of the learned hypothesis (and thereby reducing the variance of the learning algorithm). We applied three feature selection methods.

First, we employed a stopword list to eliminate words that are very common, such as "to", "open", "Microsoft" (which appears in the titles of IE and MS Office applications), "RE" and "FWD" (which appear in the subjects of email messages).

Second, we applied a simple rule-based algorithm for *stemming* English words to their roots [15]. For example, this converts "tracing", "traced", and "tracer" to the root word "trace" [15].

Third, we employed mutual information to select the $K = 200$ features with highest (individual) predictive power. Mutual information (or information gain) is one of the most common measures of relevance in machine learning [19]. It measures the reduction of entropy in the predicted class distribution $P(y|x_j)$ provided by knowing the value (i.e., present or absent) of feature x_j. Entropy is a measure of the uncertainty of a random variable. Let $\{y_i\}_{i=1}^m$ denote the set of task categories, then the mutual information of a word feature x_j is computed as

$$\begin{aligned} G(x_j) = &- \sum_{i=1}^{m} P(y_i) \log P(y_i) \\ &+ P(x_j = 1) \sum_{i=1}^{m} P(y_i|x_j = 1) \log P(y_i|x_j = 1) \\ &+ P(x_j = 0) \sum_{i=1}^{m} P(y_i|x_j = 0) \log P(y_i|x_j = 0), \end{aligned}$$

where the probabilities are estimated from the training sample using maximum likelihood estimates (i.e., simply computing the fraction of cases). The $K = 200$ features with the highest information gain are selected for inclusion in the training set.

An additional advantage of feature selection is that it speeds up the learning algorithms. For example, the TaskPredictor extracts on the order of 1000 words from the WDSs collected over a 3-month period. Our experiments show that predictive performance is maximized when the number of selected features K is in the range 100–300. This speeds up the learning algorithm and the prediction process by a factor of 3–10. As a result, our hybrid Naive Bayes + SVM classifier can make a prediction in less than 0.01 seconds, with an ordinary PC computer (*Pentium*4 CPU, 512MB).

3.3 A Hybrid Naive Bayes + SVM Classifier

The TaskPredictor applies a combination of two well-known algorithms to make its predictions.

The first algorithm is the Naive Bayes algorithm. It learns a model of the joint probability, $P(\mathbf{x}, y)$, of the input $\mathbf{x}$ and the label y, and makes its predictions by applying Bayes' rule to calculate $P(y|\mathbf{x})$. Given a test instance $\mathbf{x} = \{w_i\}_{i=1}^{|F|}$, where $|F|$ is the total number of features, we make the prediction according to the following rule:

$$argmax_y P(y|\mathbf{x}) = argmax_y \frac{P(y) \prod_{i=1}^{|F|} P(w_i|y)}{\sum_{y'} P(y') \prod_{i=1}^{|F|} P(w_i|y')}$$

Our Naive Bayes model employs an indicator (0/1) variable for each feature. We apply standard Laplace smoothing when computing the probability estimates.

The second algorithm is the linear support vector machine that has been shown to be both very fast and effective for text classification problems [12]. To apply them in our multiclass situation, we employ the one-against-one approach in which an SVM classifier is learned for each pair of classes. If there are k classes (i.e., user tasks), then $k(k-1)/2$ classifiers must be trained. To predict the class of a new case, each of these classifiers makes a prediction, and the predictions are then combined by the method of Wu et al. [18] to produce estimated probabilities $P(y|\mathbf{x})$.

Many studies in machine learning have shown that discriminative classifiers (such as SVMs) generally give higher predictive accuracy than generative classifiers (such as Naive Bayes) except at very small sample sizes. However, an advantage of generative classifiers is that they can very cheaply provide an estimate of $P(y|\mathbf{x})$ and $P(\mathbf{x})$ as we have seen above. This permits them to "know what they know" — that is, to produce a measure of the probability that they

Table 1: Datasets for Evaluating TaskPredictor.WDS (number of words is computed after stoplist and stemming)

Subject	FA	FB
# of tasks	96	81
# of WDSs	5894	4151
# of words	1202	983

Table 2: Email Datasets for Evaluation

Subjects	FA	RA	RB	SA	SB	SC	SD	SE
# of messages	459	416	244	289	869	243	458	305
# of tasks	21	23	12	9	8	14	5	15
# of features	934	721	379	613	1158	598	448	349

have seen similar data points before.

Hence, in our hybrid method, we first apply the Naive Bayes classifier to estimate $P_{NB}(y|\mathbf{x})$ and $P_{NB}(\mathbf{x})$. For TaskPredictor.WDS, we compare $P_{NB}(\mathbf{x})$ to θ and make a prediction using the learned SVM if $P_{NB}(\mathbf{x}) > \theta$. For TaskPredictor.email, we compute $\hat{p} = \max_y P_{NB}(y|\mathbf{x})$ and make a prediction using the learned SVM if $\hat{p} > \theta$. In this way, we obtain the advantages of both generative methods (that they can produce a density estimate over the input space) and discriminative methods (that they generally produce more accurate decisions).

4. EXPERIMENTAL RESULTS

We deployed TaskTracer on Windows machines in our research group and on the machines of a few willing graduate students. The users include 2 professors, 2 research staff, and 5 graduate students. We refer to them as FA and FB (the faculty members), RA and RB (the research staff), and SA, SB, SC, SD and SE (the students). All of them work in the School of EECS for the TaskTracer group except for student SC. To evaluate TaskPredictor.WDS, we required a dump of the TaskTracer database, which raises many privacy concerns. As a result, we only obtained data from FA and FB. Subject FB manually reviewed his data and cleaned it so that it contained more accurate task labels. The collected data is summarized in Table 1.

To evaluate TaskPredictor.email, we asked the participants to take a sample of their email and sort it into a new set of email folders according to their TaskTracer tasks (with a folder named "Other" for email that did not relate to any TaskTracer task). We then wrote a program that each user could run over these email folders to apply the learning algorithms and compute the performance statistics. This enabled the users to keep their email private. We obtained email data from all of the participants except FB. Table 2 summarizes the data collected for evaluating TaskPredictor.email.

4.1 Evaluation Methodology

For TaskPredictor.WDS, we adopted an on-line prediction methodology as follows. The data is sorted according to whole days. To make predictions for the WDSs in day t, we train the hybrid classifier on the data from days 0 through $t-1$. This process is started on day 1 and repeated until all data have been processed. We analyze the results based on the total number of predictions and the percentage of those that are correct.

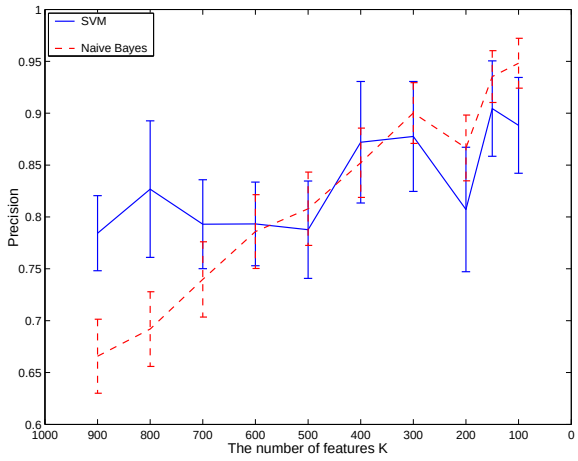

Figure 2: Precision of TaskPredictor.WDS for FA as a function of the number of selected features K. Error bars denote 95% confidence intervals.

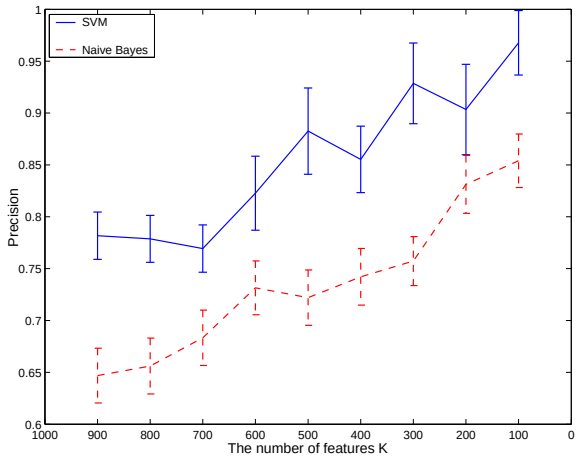

Figure 3: Precision of TaskPredictor.WDS for FB as a function of the number of selected features K. Error bars denote 95% confidence intervals.

For TaskPredictor.email, we adopted the standard training set/test set methodology, since the data was much sparser. We sorted the messages by date received and employed the first 80% for training and the remaining 20% for testing. Again, we analyze the results based on the total number of predictions and the percentage of those that are correct.

4.2 Effect of Feature Selection

We first assess the value of performing feature selection. Figures 2 and 3 show the results of feature selection for FA and FB on TaskPredictor.WDS. The horizontal axis shows the value of K, the number of features chosen by the mutual information method. The vertical axis shows the precision of the predictions. In each case, we have varied the value of the classification threshold θ to maximize the precision, subject to the constraint that θ should be large enough that the learning algorithm makes at least 100 correct predictions.

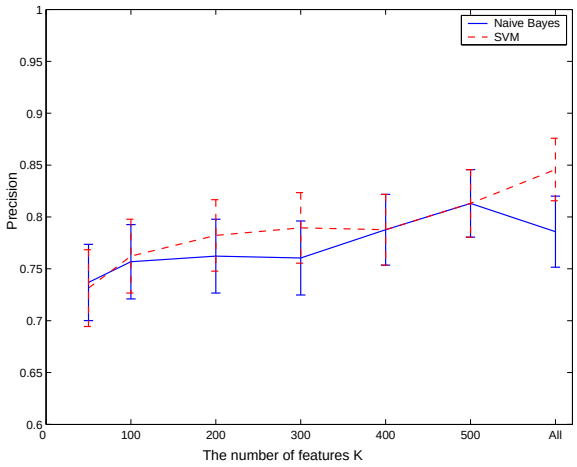

Figure 4: Precision of TaskPredictor.email as a function of the number of features K ($\theta = 0$). Error bars denote 95% confidence intervals.

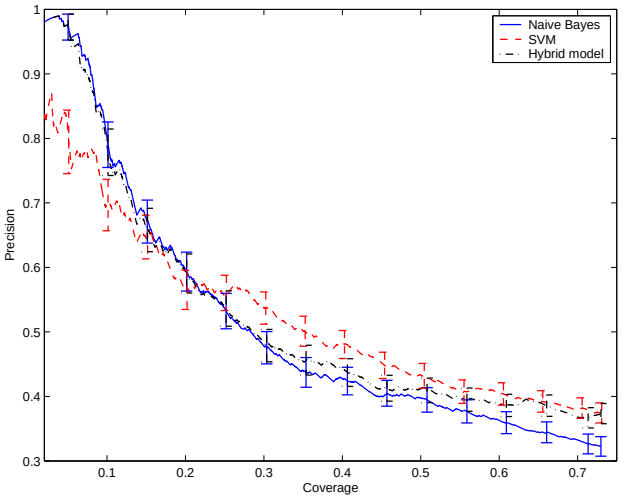

Figure 5: Precision of TaskPredictor.WDS as a function of the coverage for FA, created by varying θ.

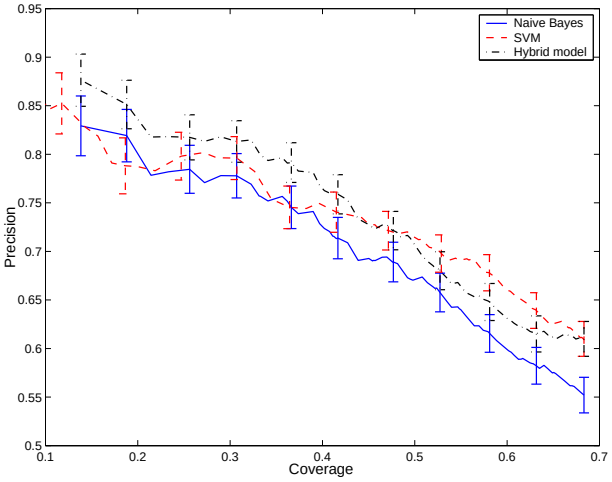

Figure 6: Precision of TaskPredictor.WDS as a function of the coverage for FB, created by varying θ.

For both FA and FB, we can see that reducing the number of features generally improves the performance of both Naive Bayes and the SVM. For FA, SVM gives higher precision than Naive Bayes when the number of features is very large, whereas Naive Bayes does better when the number of features is less than 400 (although the differences are not statistically significant). For FB, the SVM always outperforms Naive Bayes, and most of the differences are significant. Note that for FA, the best precision attained is around 93%, while for FB, the SVM does better than 95%. We set the number of features K to be 200 for both FA and FB.

Figure 4 plots the precision as a function of the number of features for TaskPredictor.email. Here, the results are quite different, perhaps because the classification threshold was set to zero, so that TaskPredictor.email was forced to make a prediction for each email message. We see that best performance is achieved when a large number of features are available. Indeed, the SVM achieved maximum performance when all features are included, while Naive Bayes attains the maximum when 500 features are available. However, none of these differences is statistically significant, so we set $K = 200$, because it makes the learning and prediction algorithms much more efficient.

4.3 Effect of the classification threshold

Now we analyze the effect of the classification threshold. Figure 5 and 6 show the precision of TaskPredictor.WDS as a function of the coverage of the algorithm. These values are averaged over all of the days of the test data. The coverage is the percentage of WDSs for which a prediction was made, and the precision is the probability that the predictions made were correct. The value of the classification threshold θ is implicit in these plots. Low values of θ correspond to high coverage, and high values of θ give low coverage. We see that as we increase θ, the precision increases very well. TaskPredictor.WDS is able to attain a precision of 80% with coverage of 10% for FA and a precision of 80% with coverage of 20% for FB.

Figure 7 plots the precision versus coverage for TaskPredictor.email. Again we observe that larger values of θ (i.e., lower coverage) give higher precision. Naive Bayes is able to attain a precision of 91% with a coverage of 66%, and the hybrid method is able to do slightly better, with a precision of 92% at a coverage of 66%. Note that the hybrid method always gives better precision for the same coverage than Naive Bayes.

4.4 Effect of the Hybrid Method

The preceding figures also show the effect of the Hybrid method. For FA (Figure 5), the Hybrid method gives precision that is essentially the same as Naive Bayes when coverage is small and the same as the SVM when coverage is larger. For FB (Figure 6), the Hybrid method gives better performance than either Naive Bayes or the SVM when coverage is small and performance similar to the SVM with coverage is larger. For TaskPredictor.email (Figure 7), we see that the Hybrid method always outperforms the Naive Bayes method at all coverage levels.

These results show that the hybrid method gives performance equal to or better than the best single method (Naive

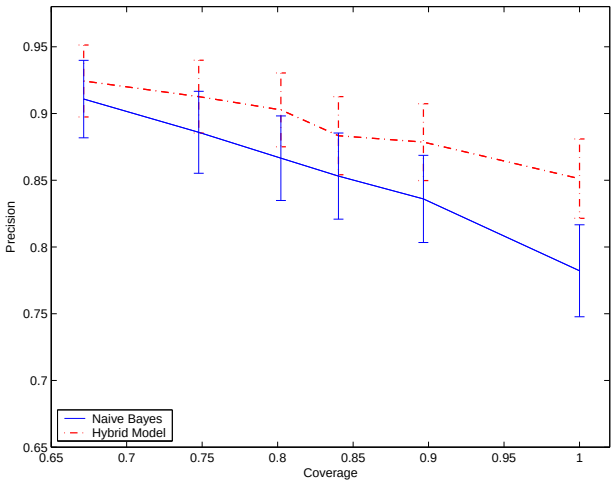

Figure 7: The precision curve of naive Bayes and the hybrid method. We adjust the thresholds so that they will make the same numbers of wrong predictions, and compare the correct ratio.

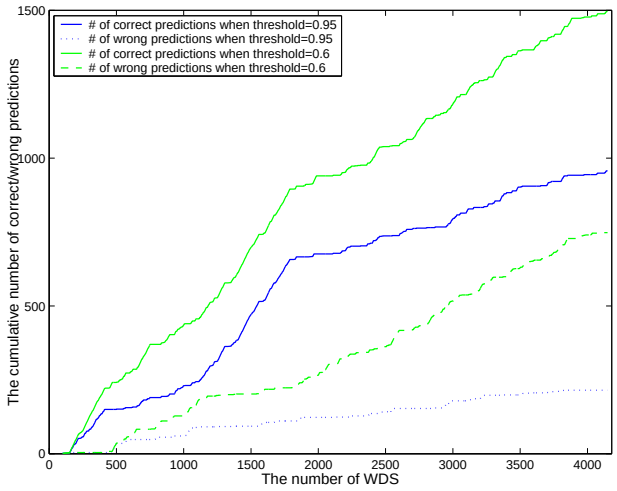

Figure 8: Cumulative correct and incorrect predictions for FB as a function of the number of WDSs processed (the hybrid model)

Bayes or SVMs). This supports our hypothesis that using a generative model to decide when to make predictions works better than using a discriminative model. This result may have significance beyond Task Prediction, and we plan to test the hypothesis over a larger set of machine learning problems to ascertain its generality.

4.5 Online Performance

It is interesting to plot the cumulative online performance of the learning algorithms over the evaluation period. Figure 8 plots the total number of correct (and incorrect) predictions for FB as a function of the number of WDSs processed, for two different settings of θ. That is, a point (x, y) is plotted at the moment when the algorithm has observed and trained on x WDSs, and it has made y correct (or incorrect) predictions.

Note that there are several times in which the error curves (the lower two curves) take sudden upward jumps. These correspond to periods when the user starts a new task. The new task may confuse TaskPredictor because 1) TaskPredictor has seen very few training examples for the task and 2) the user may access resources of other tasks in order to initialize the new task (e.g., by copy and edit). This suggests that it may be appropriate to disable TaskPredictor.WDS for some period of time after a new task is initialized. It also shows that even the Hybrid method is not able to identify all cases where it should *not* make a prediction.

5. RELATED WORK AND DISCUSSION

There have been many attempts in machine learning to study the problem of task recognition and prediction. Researchers usually convert this inherently sequential problem into an ordinary supervised learning problem through the design of appropriate features [2]. Many of the applications are based on probabilistic models. The *Lumière* project centers on harnessing probability and utility to provide assistance to computer users [9]. Lumière applies Bayesian network user models to infer a user's needs by considering a user's background, actions, and queries. Recently, Horvitz et al.[11, 10] have been studying the problem of interruption. They learned a dynamic Bayesian network to model the user's attentional focus and predicted the cost of interrupting the user. Intel Research has developed a toolkit called the Probabilistic Activity Toolkit (PROACT) [14]. They tried to infer the user's activity from the objects involved in the activity. PROACT's activity model is restricted to linear sequences of sub-activities which provide annotated object information. The model is very similar to a hidden Markov model.

Many researchers have previously studied the problem of email foldering, which is defined as predicting into which of the user's email folders an email message will be filed. In 1996, William Cohen proposed RIPPER, a rule based learning algorithm, and applied it to email classification [5]. RIPPER employs information-gain feature selection during rule construction. For email classification, it uses features based on the recipients, subject line, and email body. IBM developed MailCat, which later became the SwiftFile component of Lotus Notes. It helps the user to select folders for incoming emails [16]. The MailCat team adopted AIM, a TF-IDF like algorithm, to do the classification job. Brutlag and Meek [4] compared three different text classification algorithms in the email domain (SVM, TF-IDF, naive Bayes), and they reported that TF-IDF performed better than the other two algorithms for sparse folders. Some recent research has been done on the Enron email dataset, which is a large email collection containing 0.5M messages and 150+ users, who are mostly senior managers of Enron [3, 13].

Most previous research shows 50%-80% accuracy if the classifier makes a prediction for all the email messages. The accuracy can vary a lot from one user to another, even within the same experimental study. This is probably because some users create email folders for different purposes. Our email prediction problem is not quite the same as the email foldering problem. Our goal is to predict which on-going activity should be associated with the email message. By encouraging users to think in terms of activities, we may be leading them to define better email groupings than the traditional

folder hierarchies that most users construct. Indeed, a frustrating aspect of email foldering research has been that the folders may be highly inconsistent and have very different scopes.

Like other groups, we treated the task prediction problem as a traditional supervised learning problem, and we ignored the sequential aspect of the problem. There may be an opportunity to improve the results reported in this paper by extending our methods to handle the sequential correlations in the data. Specifically, if the user begins a task, he or she is likely to continue working on that task for many minutes and even hours. The user is likely to come back to the task over the following days and weeks. We may be able to incorporate this either within a Markov framework (e.g., using hidden Markov models, dynamic Bayesian networks [17], or hidden Markov support vector machines [1]). However, initial experiments showed that while these models give marginally better performance, the computational cost of learning them is prohibitive. An attractive alternative is to incorporate these sequential relationships into standard supervised learning methods through the addition of other features that capture the amount of time that has elapsed since the user last worked on each task.

Similarly, we may want to incorporate some form of recency weighting into the algorithms to put more weight on recent WDSs and email messages either globally or within each task. This has given improved accuracy and robustness in some of our other work.

This paper has presented two learning systems for predicting the current activity of the user. The TaskPredictor.WDS system predicts the user's current TaskTracer task based on the title and document pathname (or URL) of the window currently in focus. The TaskPredictor.email system predicts the current task based on the sender, recipients, and subject of an email message.

We demonstrated that three machine learning techniques gave improved performance with these systems: 1) feature selection via mutual information, 2) a threshold for making classification decisions, and 3) a hybrid approach in which a generative model (Naive Bayes) is first applied to decide whether to make a prediction and then a discriminative model (linear support vector machines) is applied to make the prediction itself. The experiments show that the hybrid method gives slightly better performance than either Naive Bayes or SVMs alone. These three techniques give us task predictors that are sufficiently accurate to be useful. Our next challenge is to incorporate them into the TaskTracer system in a way that improves the usability and benefits of the TaskTracer system.

Acknowledgments

This project was supported in part by the National Science Foundation under grant IIS-0133994 and by the Defense Advance Research Projects Agency under grant no. HR0011-04-1-0005 and contract no. NBCHD030010. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the Defense Advanced Research Projects Agency, or the Department of Interior-National Business Center. The authors thank the members of the TaskTracer team for all of their work designing, implementing, and testing the system and for contributing their TaskTracer data to this study.

6. REFERENCES

[1] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *Proc. of ICML-03*.

[2] S. Andrews, L. Cai, D. Gondek, A. Greenwald, D. Grollman, A. M. Jonsson, K. Hall, M. Lease, B. Ng, J. Raiti, V. Sweetser, and J. Turner. Astrology: the study of astro teller. In *ICML04 Workshop Physiological Data Modeling - A Competition*, 2004.

[3] R. Bekkerman, A. McCallum, and G. Huang. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. Technical Report IR-418, CIIR, 2004.

[4] J. D. Brutlag and C. Meek. Challenges of the email domain for text classification. In *Proc.of ICML-00*.

[5] W. W. Cohen. Learning rules that classify e-mail. In *Proc. Of the 1996 AAAI Spring Symposium in Information Access*, 1996.

[6] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proc. of IUI-05*.

[7] T. Fawcett and F. Provost. Activity monitoring: notice interesting changes in behavior. In *Proc. of KDD-99*, 1999.

[8] K. Haigh and H. A. Yanco. Automation as caregiver: a survey of issues and technologies. In *AAAI workshop on Automation as Caregiver*, 2002.

[9] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of UAI-98*, 1998.

[10] E. Horvitz, A. Jacobs, and D. Hovel. Learning and reasoning about interruption. In *Proc. of ICMI-03*.

[11] E. Horvitz, A. Jacobs, and D. Hovel. Attention-sensitive alerting. In *Proc. of UAI-99*, 1999.

[12] T. Joachims. *Learning to Classify Text Using Support Vector Machines*. Kluwer Academic Publishers, 2001.

[13] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Proc. of ECML2004*, 2004.

[14] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, and D. Hahnel. The probabilistic activity toolkit: Towards enabling activity-aware computer interfaces. Technical Report IRS-TR-03-013, Intel Research Lab, Seattle, WA, 2003.

[15] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[16] R. Segal and J. Kephart. Mailcat: an intelligent assistant for organizing e-mail. In *Proc. of the Third ICAA*, 1999.

[17] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden markov probability models. *Neural Computation*, 9(2):227–269, 1997.

[18] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. In *Advances in NIPS 16*.

[19] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of ICML-97*.