Active EM to Reduce Noise in Activity Recognition

Jianqiang Shen
1148 Kelley Engineering Center
School of EECS, Oregon State University
Corvallis, OR 97331, U.S.A.
shenj@eecs.oregonstate.edu

Thomas G. Dietterich
1148 Kelley Engineering Center
School of EECS, Oregon State University
Corvallis, OR 97331, U.S.A.
tgd@eecs.oregonstate.edu

ABSTRACT
Intelligent desktop environments allow the desktop user to define a set of projects or activities that characterize the user's desktop work. These environments then attempt to identify the current activity of the user in order to provide various kinds of assistance. These systems take a hybrid approach in which they allow the user to declare their current activity but they also employ learned classifiers to predict the current activity to cover those cases where the user forgets to declare the current activity. The classifiers must be trained on the very noisy data obtained from the user's activity declarations. Instead of asking the user to review and relabel the data manually, we employ an active EM algorithm that combines the EM algorithm and active learning. EM can be viewed as retraining on its own predictions. To make it more robust, we only retrain on those predictions that are made with high confidence. For active learning, we make a small number of queries to the user based on the most uncertain instances. Experimental results on real users show this active EM algorithm can significantly improve the prediction precision, and that it performs better than either EM or active learning alone.

ACM Classification: I.2.1 [Artificial Intelligence]: Applications and Expert Systems. - Office automation.

General terms: Design, Human Factors, Experimentation.

Keywords: Intelligent interface, machine learning, noise, Expectation-Maximization, active learning.

INTRODUCTION
Several groups are developing "activity-aware" desktop environments that model the desktop user as switching among a collection of ongoing activities. For example, the TaskTracer system [28] allows the user to define a set of activities such as "teach CS534", "IUI-2007 paper", "write NSF proposal" and so on. TaskTracer captures a wide range of desktop events (file open, save, email send, web page visit, cut/paste, etc.). It then builds and maintains a database of associations between each activity and the "resources" (i.e., files, folders, web pages, email messages, email addresses) that are accessed when working on that activity. This set of associations is then applied to help the user in several ways:

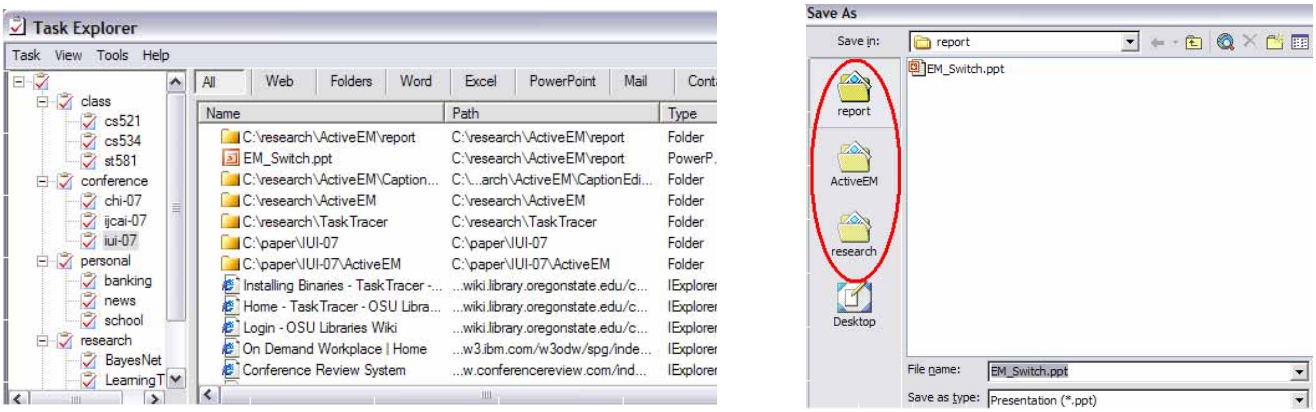
Finding and Accessing Resources. Multi-tasking users are frequently interrupted. When the user returns from an interruption (after a few minutes, hours, or days), the user can obtain a display of the "resources" (files, folders, web pages, etc.) associated with the activity (see Figure 1(a)). This reminds the user of what the user was working on before the interruption. Double clicking on a desired resource (e.g., a Word file) launches the appropriate program. Having all resources in one place is much more efficient than searching in desktop search or going through various applications (email, Windows Explorer, IE Favorites, etc.) looking for the right resource.

Folder Prediction. When the user initiates an Open or SaveAs, TaskTracer estimates the probability $P(f)$ that the user will want to open/save in folder f and initializes the Open/Save dialogue box in the best predicted folder. Specifically, it computes the set of three folders that jointly minimize the expected number of clicks to reach the target folder (see Figure 1(b)).

The key to providing these services is to correctly associate each activity with the relevant resources. One way to do this is to require the users to explicitly declare their current activity, and then associate each accessed resource with the declared activity (as in the UMEA system [15]). TaskTracer supports this by providing a drop-down box that makes it easy for the user to declare the current activity, either by selecting with the mouse or by typing the name of the activity (with auto-completion). The drop-down box is always attached to the window in focus. This explicit declaration of the activity makes sense when the user is returning from an interruption and wants to bring up the list of relevant resources. However, this approach fails when the user is interrupted (e.g., by a phone call, IM). The user typically changes documents, web pages, etc. without remembering to first inform TaskTracer. In such cases, we would like TaskTracer to automatically detect the activity switch and correctly associate the new resources with the correct activity.

The learning challenge is to predict the current TaskTracer activity of the user based on their current desktop actions (opening files, visiting web pages, reading emails, etc.). As

IUI'07, January 28–31, 2007, Honolulu, Hawaii, USA.



(a) Task Explorer (b) Folder Predictor

Figure 1: The TaskExplorer lists all resources related to the current activity; the Folder Predictor provides 3 predicted folders in the "places bar" at left

training data, we have available all of the previous resources that were accessed by the user and the name of the corresponding activity that was declared by the user. Hence, this can be formulated as supervised learning with very noisy labels. This paper describes learning methods for this noisy learning task.

Our basic approach is to first train a classifier using the noisy labels (i.e., treating them as correct). This classifier then serves to initialize an unsupervised learning process in which the Expectation-Maximization (EM) method [8] is applied to maximize the likelihood of the data (ignoring the class labels). We combine this EM approach with *pool-based* active learning [18] to reach even higher precision. Our active learner requests from the user the true class labels for instances about which it is most uncertain. Because each query is considered as an expensive operation, we only make a limited number of queries. We select queries based on a heuristic criterion which measures the entire uncertainty that the instance value contributes to the training set.

The paper is organized as follows. First, we define the prediction task and the input features that we employ. Second, we describe our solution to reduce the noise in activity recognition – an active EM algorithm, which combines the Expectation-Maximization (EM) method [8] and an active learning strategy. We then present an experimental study of its effectiveness. The paper concludes with a discussion of the results.

THE ACTIVITY PREDICTION TASK

TaskTracer collects events from MS Office 2003, MS Visual .NET, Internet Explorer, and the Windows XP operating system. Then various TaskTracer components provide services to the user based on this information. From this stream of raw events, TaskTracer extracts a sequence of *Window-Document Segments* (WDSs). A WDS is a maximal contiguous segment of time in which a particular window has focus and the name of document in that window does not change. Hence, each time the user switches from one window to another or invokes SaveAs, this begins a new WDS.

TaskTracer constructs a "bag of words" representation of each WDS by extracting words from the window title and the file pathname or URL. Words are stemmed [26] and words appearing in the "stopword list" are removed. We will denote the resulting bag of words by $\mathbf{x} = (x_1, \ldots, x_j, \ldots, x_n)$, where x_j is an indicator variable that is 1 iff the j-th word appeared in the WDS bag of words. Let $\mathbf{x}_i$ denote the training bag-of-words for the i-th WDS and y_i be the user's (noisy) declared activity for that WDS. Given a set of training examples, we compute the mutual information between each feature x_j and the class label y and select the 200 features with highest (individual) mutual information. Our previous work [28] has shown that using mutual information can significantly improve the prediction precision.

Note that the same bag of words can be observed multiple times in our data for two reasons. First, each time the user returns to an application window after visiting another window, TaskTracer creates another $\mathbf{x}$ from the same window title and pathname or URL. Second, there are relatively few words in each $\mathbf{x}$, so it is possible that after stemming and stopword removal, two webpages on the same website or two documents in the same folder will yield the same bag of words.

To collect the labeled training data, we rely on the user to indicate the activity by selecting the activity name from a special drop-down box, and it's likely that the user sometimes will forget to do this, especially in a busy time period. This will make the labeled data quite noisy and lead to poor predictions. To reduce the noise in the training data, we wrote an application called the "Event History Editor". A screenshot of this Editor is shown in Figure 2. This application lists the event records in a specified time range. The user can review the activity log and correct the labeling mistakes. However, reviewing these records is a painful and mistake-prone job, considering the large volume of records. Only a few people have the stamina to do this. Hence, we need some automatic mechanisms to relieve the user from this burden.

LEARNING ALGORITHMS

When we train on the noisy data, we believe that the user's labels could be wrong. Hence, we adopt the Expectation-

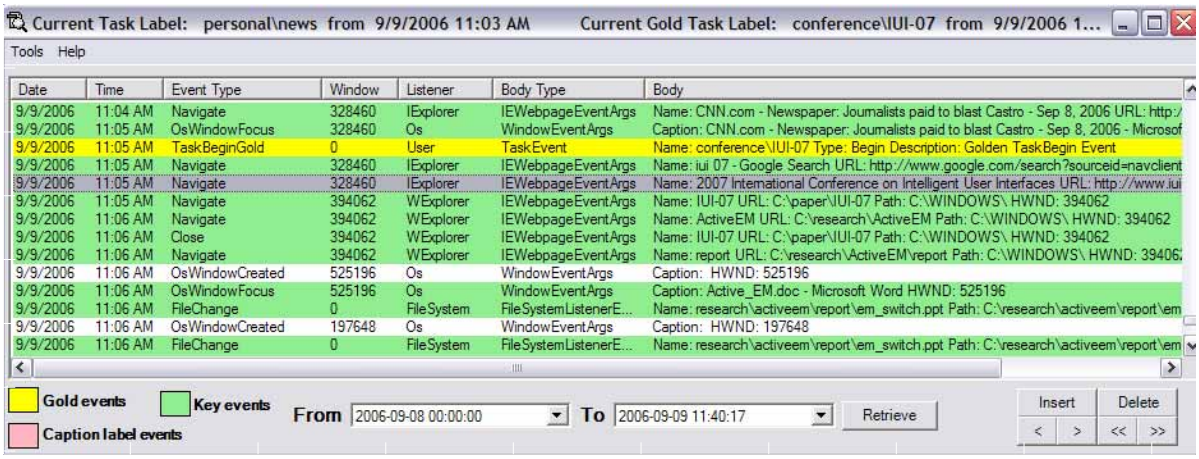


Figure 2: The screenshot of Event History Editor, which is used to relabel the user's activity. "Caption label events" are TaskBegin events generated by selecting the activity name from a drop-down box at run time, and "Gold events" are the corrected TaskBegin events given by the user at post-processing time. "Key events" are events useful for training.

Maximization (EM) method [8], an unsupervised learning technique that can handle incomplete and missing data. In this case, EM can be viewed as down-weighting the user's labels when those labels have low probability according to the learned model.

To make EM more robust, we probabilistically relabel the training instances only when the relabeling is confident. We also augment EM with active learning by issuing queries on the most uncertain instances. In this paper, we employ naive Bayes as our basic classifier, but the approach can also be applied to other probabilistic classifiers.

Selective EM

EM is a class of iterative algorithms for maximum likelihood estimation in problems with incomplete data [8]. Here, we initialize EM by training an initial classifier h based on the noisy instances. Then EM repeatedly performs a two-step procedure: in the *E-step*, it uses h to compute $P(y|\mathbf{x})$, the probability distribution over the predicted labels for each instance $\mathbf{x}$. This can be viewed as "relabeling" the training data using the predictions from h. However, the labels are probabilistic: instead of assigning a single label to each $\mathbf{x}$, EM assigns the probability distribution $P(y|\mathbf{x})$. Then, in the *M-step*, EM fits a new maximum a posteriori classifier h using the predicted label probabilities as weights. The idea is to initialize EM "onto the right hill", and then hill-climb to the top.

By exploiting word co-occurrence information contained in the noisy data, EM can help the classifier recover from incorrect labels. Even if a resource (represented as a feature vector) was uniformly accessed by different activities because the user forgot to change activities, such noise can still be reduced if this feature vector contains some discriminative features. For example, suppose that, because of mislabeling, a document named "*Events Listener*" was labeled roughly the same number of times with the activity "project TaskTracer" and the activity "project Bugs". Suppose that the word "Listener" appeared only in this document name and therefore, it has no predictive value (because the document appeared an equal number of times in both activities). But fortunately, the word "Events" appeared in other WDSs that were labeled with "project TaskTracer". Therefore, in the initial fitted model, EM learns that "Events" is discriminative. This allows it to correctly predict that "*Events Listener*" is associated with "project TaskTracer" and therefore to correct some of the mislabeled WDSs. In the M step, it will therefore be able to learn that the word "Listener" provides discriminative evidence for "project TaskTracer". In effect, the discriminative power of "Events" was propagated to "Listener". This spreading of discriminative power continues until convergence.

This approach is similar to previous applications of EM in semi-supervised learning [23, 25, 21]. These studies have shown that it can be unreliable and sometimes gives poor results. Our preliminary study also shows that EM sometimes decreases prediction precision. A possible explanation is that noisy labels still contain useful information, but the approach described above ignores the noisy labels after the initialization step. These noisy labels might be especially useful when the EM classifier predicts a nearly uniform probability distribution for the class labels. In such situations, using the predicted probabilities as weights may be worse than using the original noisy labels.

We therefore adopt a mixed strategy in which we "relabel" training examples when the EM classifier is highly confident, but retain the original noisy labels when it is not confident. Let the initial classifier's prediction accuracy for bag-of-words $\mathbf{x}_i$ be c_i. We can crudely approximate EM as a process of complete relabeling (as in Gibbs sampling). In this case, the predicted labels will be wrong with probability $1 - c_i$. If the user's label for $\mathbf{x}_i$ is wrong with probability ϵ_i, it would be better to retain the user's label if $1 - c_i > \epsilon_i$. Of course, we don't know either c_i or ϵ_i. Instead, we introduce a tunable confidence threshold α. Let $\tilde{p}_i$ be the probability of the most likely class label $P(y|\mathbf{x}_i)$ as predicted by the current classifier during EM. If $\tilde{p}_i < \alpha$, then in the M step, we

Procedure ACTIVEEM(S, α, m, k)

Input: S, the data set ; α, the threshold of EM ; m, the maximal iteration number ; k, the maximal query number.

Initialize classifier h;
do ()
  for each instance i in S, assuming its feature vector is $\mathbf{x}_i$
    if (a query has provided the label $\widetilde{y}_i$ for $\mathbf{x}_i$) predict $\widetilde{y}_i$ with $P(\widetilde{y}_i|\mathbf{x}_i) = 1$;
    else compute the posterior probability distribution $v_i$of $\mathbf{x}_i$ using h;
    if $(\max_y P(y|\mathbf{x}_i) > \alpha)$ update the training weight with the probabilities;
    else keep using the original label;
    Compute $q(\mathbf{x}_i)$ from v_i;
  if (the number of queries until now $< k$)
    Choose the instance **x** with the highest $q(\mathbf{x})$, and ask the user for the label of **x**;
    if (the user indicates a label for **x**) update the labels of all instances with value **x**;
    else add all features of **x** to the stopword list;
  Retrain the classifier h;
until (convergence or the number of iterations is $\geq m$)
return (h);

Figure 3: The Active EM algorithm, which combines active learning with selective EM.

employ the user's original noisy label y_i. Otherwise, we follow the normal EM approach of using the weighted labels $P(y|\mathbf{x}_i)$. Our reasoning is that when $\tilde{p}_i$ is large, the prediction is very likely to be correct [28], so it is likely that $1 - c_i < \epsilon_i$. We call this the *Selective EM* algorithm. It converges rapidly (usually in fewer than 10 iterations).

Active Learning

Selective EM is able to repair noisy labels if the fitted naive Bayes model is very confident. But it cannot recover from noisy labels when the fitted model is uncertain.

Active learning is a methodology for choosing queries to ask the user, and then using the results of those queries to constrain learning [1]. Active learning has been studied in the context of many natural language processing applications. *Pool-based* active learning for classification was introduced by Lewis and Gale [18]. The learner has access to a pool of unlabeled data, and it can request the true class labels for a certain number of instances in the pool. The main issue with active learning is to find a way to choose good queries from the pool. In many settings, an active learner is usually designed to select instances that, when labeled and incorporated into training, will most reduce the expected prediction error over the distribution of future instances [7, 19].

When the training set is large, the computation to find the statistically optimal query is expensive. For an interactive intelligent desktop application, we need a very efficient method. The simplest one is to randomly pick some unqueried bags of words, and ask for labeling. We can also use *Error Sampling* to select the query, which prefers the instance with a small absolute difference between the estimated class probabilities of the two most likely classes [6, 27]. But neither of these methods takes into account the fact that the same bag of words can be observed multiple times in our data. In this paper, we adopted the following procedure, which gives superior results. Given bag-of-words $\mathbf{x}_i$, we define the uncertainty about $\mathbf{x}_i$ as:

$$q(\mathbf{x}_i) = n_i H(y|\mathbf{x}_i) = -n_i \sum_y p(y|\mathbf{x}_i) \log p(y|\mathbf{x}_i), \quad (1)$$

where n_i is the count of times that bag-of-words $\mathbf{x}_i$ occurred in the training set and $H(y|\mathbf{x}_i)$ is the entropy of the predicted label probability distribution. Entropy is a measure of the amount of uncertainty about an event associated with a given probability distribution. If the probabilities of all tasks are close and no one predominates, then our uncertainty (and, hence, the entropy) is maximal. We take into consideration n_i, because unlike most text classification problems, our bags of words can be repeated many times in the training data. Hence, $q(\mathbf{x}_i)$ reflects the entire uncertainty that bag-of-words $\mathbf{x}_i$ contributes to the training set. Note that we can compute the q values efficiently as a byproduct of the EM computation. We pick the bag-of-words with the highest q value, and ask the user to give the real label for it.

One potential problem with active learning is that sometimes the user finds it difficult to assign a label to the query. In TaskTracer, this occurs because some bags of words do not contain any informative words. For example, when the user first starts up Internet Explorer, the default title is "*about:blank - Microsoft Internet Explorer*". Obviously this provides no information about the user's current activity. We address this by allowing the user two choices in response to our query: (a) the user can give the label of the correct activity or (b) the user can indicate that the correct label cannot be determined because the words are not informative. In the latter case, we add all of the words from the bag of words to the stopword list. Our "stopword list" is therefore the set of all words that the system believes are uninformative. It initialized with some very common words, such as "of", "Microsoft", etc., and the system will add more uninformative words into it.

This allows us to use active learning for feature selection as well as to obtain labels for uncertain data points.

Combining EM and Active Learning

Combining active learning with selective EM can reach better performance than either selective EM or active learning alone. Active learning can improve the accuracy of selective EM by making informative queries, and selective EM can help active learning choose a better query by using its improved classifier. The entire algorithm, Active EM, is described in Figure 3.

To limit the burden placed on the user, we only allow a fixed number k of queries. A query on the instance with the highest q value will be made if we have not exceeded the maximal number of queries. We can reach higher precision by allowing more queries, but at higher cost to the user.

To avoid selecting an instance whose value has already been queried, after the query on value $\mathbf{x}_i$ has been made, we attach a mark to all instances whose feature vectors are equal to $\mathbf{x}_i$. When making predictions, instances that were assigned labels as the result of a query will be predicted to have these labels with probability 1. Instances indicated as uninformative will not be used for retraining, since all of their features are in the stopword list. In each iteration of Active EM, before retraining the classifier, we re-select the 200 features with the highest mutual information with the best known label (i.e., either the label obtained from a query or the original noisy label), and we do estimation based on these new features.

EXPERIMENTAL RESULTS

We deployed TaskTracer on Windows machines in our research group. To evaluate the various algorithms, we need data for which we have both the original noisy labels and the true labels. To obtain the true labels, we implemented the "Event History Editor" described above. This allows the user to review and relabel the collected data. We then train with the noisy labels, but evaluate the results with the real labels. Because only a few people have the stamina to do this relabeling, we only obtained two datasets, which we will refer to as SA and FB[1]. Dataset SA records 2.5 months of activities, including 8 distinct activities, 512 distinct words and 4407 instances (including duplicates). Dataset FB records 2 months of activities, including 51 distinct activities, 781 distinct words and 3138 instances. On average, there are approximately two instances of each bag of words.

Procedure. We adopted an on-line learning methodology as follows. The data is sorted according to time. To evaluate each algorithm, we process the training data one day at a time starting with the day that is 25% of the way through the data set. To test an algorithm on day t, we train the classifier on the data from days 0 through $t-1$ and then measure the accuracy on day t. This process is repeated until all data have been processed. The final reported result is based on the performance averaged over all of these days (after the first 25%). The initial 25% period gives the algorithm a "running start" before it has to be evaluated.

The Active EM algorithm was allowed 4 queries per day, which it chooses at the end of each day. We did not deploy Active EM on the users' desktop machines. Instead, we simulated their answers as follows. We first employed the *entire* data set to compute the 200 features with the highest mutual information. We call this set of features Ω, and we treat these as the true set of relevant features. When Active EM makes a query for an instance that contains *no* features from these top 200, we answer that the query is "uninformative"; otherwise we answer with the real label (as provided by the user via the Event History Editor).

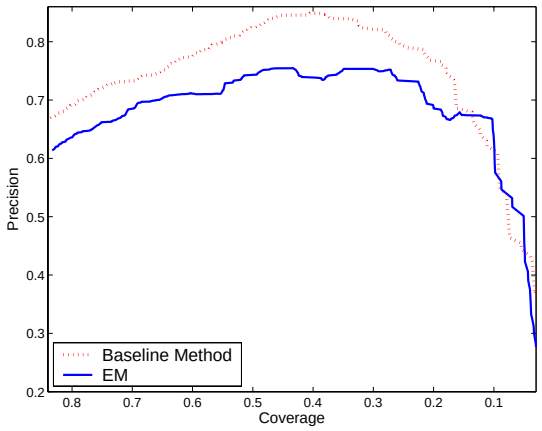


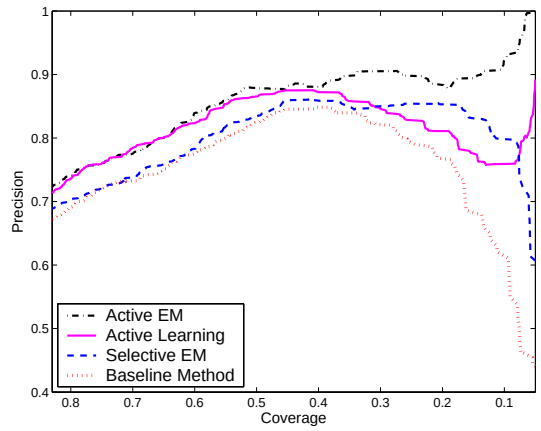
Figure 4: Precision of EM as a function of coverage for SA, created by varying the decision threshold θ.

When making predictions, we employ a rejection threshold, because in this application, an incorrect prediction is worse than no prediction. This is because if a prediction is made, we plan to have the TaskTracer system pop up a "balloon" message telling the user "TaskTracer thinks you have changed your activity; click here", which interrupts the user and asks for confirmation of the activity change.[2] The reject decision is made as follows: let $\hat{y}_i = \arg\max_y P(y|\mathbf{x}_i)$ be the label with the highest predicted probability, and let $p_i = P(\hat{y}_i|\mathbf{x}_i)$ be its predicted probability. If $p_i > \theta$ (where θ is a specified threshold), then the system makes a prediction; otherwise, it rejects the instance and remains silent (i.e., does not interrupt the user). We consider two metrics in evaluating the algorithms: *coverage* and *precision*. Coverage is the number of predictions divided by the number of instances, and precision is the number of correct predictions divided by the number of predictions issued.

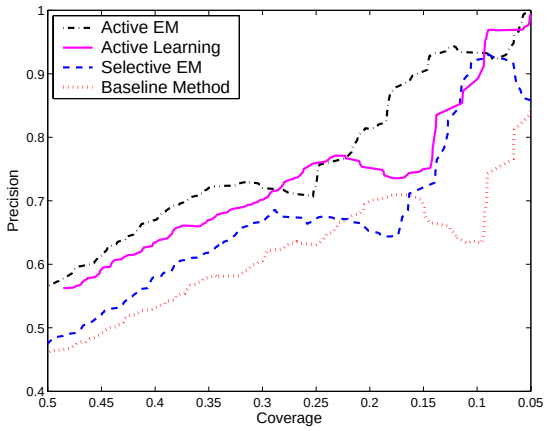
We map out a precision-coverage curve by systematically varying θ. High precision is much more important than high coverage, because we do not need high coverage to have a useful activity predictor. The system works in a "keyframe" way: if we can accurately predict the ongoing activity at enough key time points, then we can provide value between these points. Wrong predictions are worse than no prediction. Thus low coverage-high precision is the important case to optimize.

[1] Available at http://www.cs.orst.edu/~shenj/TT_Data

[2] Our actual interruption algorithm will also take into account what the user is currently doing before deciding to interrupt. It will also combine predictions from multiple WDSs and other sources of information before making this decision.

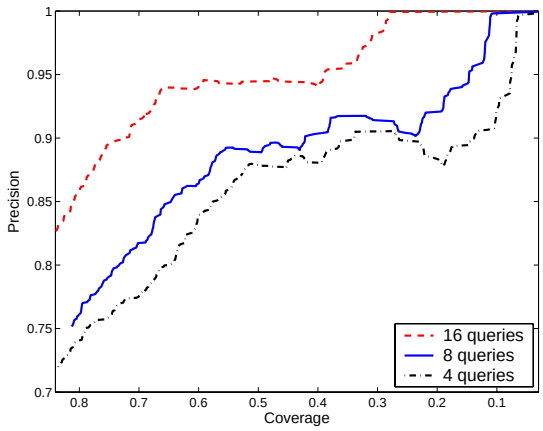


(a) SA

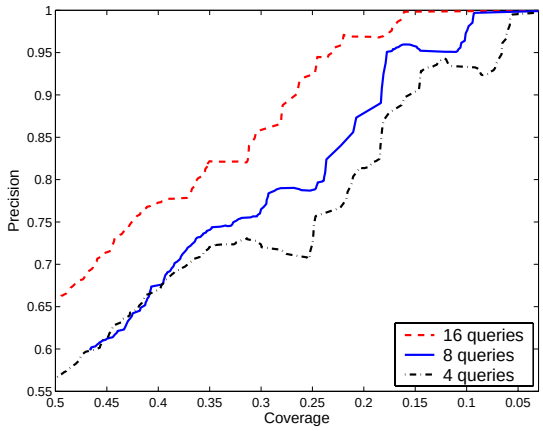


(b) FB

Figure 5: Precision of different learning methods as a function of the coverage, created by varying θ.



(a) SA



(b) FB

Figure 6: Precision of Active EM as a function of coverage, created by varying θ.

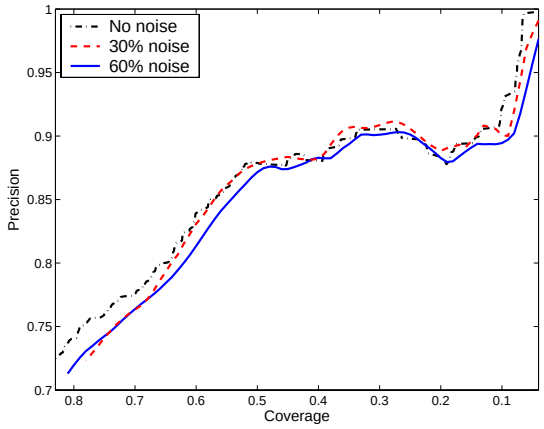
Simple EM can decrease the peformance. As we have mentioned, EM sometimes decreases, rather than increases, prediction precision. Figure 4 presents the results of our experiments for participant SA. The Baseline Method fits a naive Bayes model to the original noisy data. The EM method always uses the predicted probabilities as training weights in the *M-step*. From the plot, we see that at virtually all coverage levels EM gives precision less than or equal to the baseline method. It performs particularly poorly at medium and high coverage levels. We therefore excluded standard EM from the remaining comparisons.

Comparing Active EM with other methods. Figure 5 presents the results of our experiments for our two participants. The methods are Active EM, Active Learning, Selective EM, and the Baseline Method (Naive Bayes). Active EM is the algorithm described in the preceding section. It employs a confidence threshold of $\alpha = 0.85$ and asks 4 simulated questions per day. This α value was chosen based on previous research showing that 0.85 gives a good tradeoff between precision and coverage [28]. For the experiments in this paper, the results do not vary significantly for α values between 0.8 and 0.97. A very large α will give similar results with the baseline method, since EM will just use the original noisy labels for training. A separate set of experiments (data not shown) found that when α is smaller than 0.8, the performance begins to decrease. When $\alpha = 0$ (equivalent to non-selective EM), precision decreases by approximately 20%.

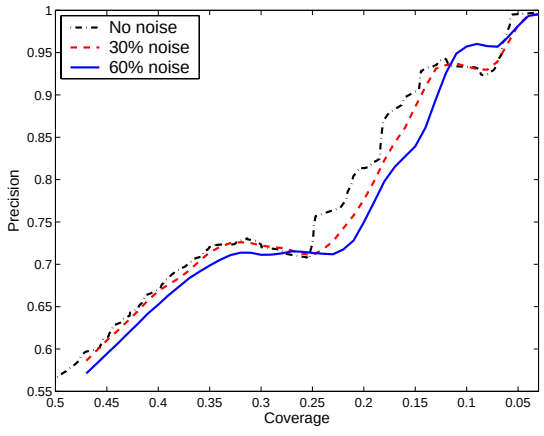
The Active Learning method was defined as follows. In each iteration, it fits a naive Bayes model to the training data (as updated by the query results) without using EM. It computes queries and retrains the classifier exactly like Active EM. The Selective EM method applies the α threshold to control EM, but does not make any simulated queries to the user. The Baseline method is simply naive Bayes fit to the original noisy data.

Figure 5 shows that for most levels of coverage, Active EM is better than Active Learning, which is better than Selective EM, which is better than the Baseline Method. At higher coverage levels, Active Learning provides a larger performance improvement than EM, but at low-to-moderage coverage levels, EM can provide a very large benefit.

With only 4 queries every day, Active EM can improve the precision over the Baseline by more than 10 percentage points. This shows that Active EM is a very effective method for reducing the noise in the training data. Of the queries that



(a) SA



(b) FB

Figure 7: Precision of the Active EM given the coverage, created by varying θ.

are made during the experiment, 21.1% are "uninformative" for SA and 43.7% are "uninformative" for FB.

Effect of more queries. Figure 6 shows what happens if we increase the number of queries allowed per day. These results show that more queries lead to better performance. With 16 queries every day, Active EM attains an additional 10 percentage points of precision. This is a nice characteristic: it allows the user to increase the number of queries if the user is not satisfied with the performance, until a satisfactory level of precision is reached.

Sensitivity to user errors in the "uninformative answer". As stated above, we simulated the "uninformative answer" using the Ω feature set. This methodology assumes that the user can determine the informativeness of the query perfectly. How sensitive are the results to this assumption?

To explore this question, we performed an additional experiment. Let the entire word set be Σ. To simulate an imperfect user, we modified Ω by randomly choosing features from Ω and replacing them with features chosen randomly from $\Sigma - \Omega$. To reduce the variance, we repeated the experiments 10 times and plotted the average results in Figure 7. With 60% of the globally informative features replaced by uninformative ones, the performance is decreased slightly. This shows that the method is robust to errors in the user's judgement of what features are uninformative. It is also possible that users, because they know more about their own activities than may be revealed in the training data, may be able to make better judgments of informativeness than we did in our automated experiment. Hence, the performance of Active EM could even be better than is shown in our experiments.

RELATED WORK

Many researchers have tried to apply machine learning to study the problem of activity recognition and prediction. The *Lumière* project applies Bayesian network user models to infer a user's needs by considering a user's background, actions, and queries [11]. Recently, Horvitz et al. [12, 13] have been employing dynamic Bayesian networks to study the problem of interruption. The PROACT project at Intel Research [24] tried to infer the user's activity from the objects involved in the activity. Mitchell et al. [20] use a clustering approach to automatically acquiring the knowledge base about the user's activity by analyzing the raw contents of the workstation. There is also a growing body of work on classifying email to activities. For example, Kushmerick and Lau's activity management system [17] uses text classification and clustering to examine email activities that represented structured business processes; Dredze et al. [9] investigated two different approaches to the problem of email activity classification, based on the people involved in an activity and the contents of messages in the activity. In this paper, we are focusing on the "noise" problem arising in activity recognition. The "noise" problem is common in these activity recognition problems and might make the learning approach fail. Our active EM algorithm only needs a little extra effort from the user to improve the recognition precision significantly.

There have been many attempts in machine learning to study the problem of learning in the presence of random classification noise including methods based on outlier detection, voted ensembles, and trained filters [5]. Noise-tolerant learning algorithms have also been studied within the standard PAC model [2, 16, 3]. These algorithms assume that there is an oracle that can cheaply generate as many instances as we want. With this assumption, these algorithms can learn to the desired precision. However, in the real world, the size of the training set is usually fixed, and it is very expensive to get additional training instances. Instead, our active EM provides a practical way to efficiently correct bad training labels and improve the precision of the classifier.

Combining labeled and unlabeled data in text categorization has received much attention. The approaches include applying Expectation-Maximization to estimate the parameters of a generative model [23], performing transductive inference for support vector machines to optimize accuracy [14], and training two classifiers using separate views of the same data [4]. Our Selective EM was inspired by the work on *self-training* [22]. Self-training initially builds a single classifier using the labeled training data. Then it labels the unlabeled data and converts the most confident instance of each class into a labeled training example. This iterates until all the unlabeled instances are given labels.

Active learning is another attempt to reduce human effort in labeling training data. The main issue is how to select a query. Pool-based active learning has access to a pool of unlabeled data and can request the true labels for some instances in the pool [18]. Query-by-committee [10] trains an ensemble of classifiers and then chooses data points where those classifiers most disagree to be the queries. Methods for reducing the integrated mean squared error, based on the statistical design of experiments have also been studied [7]. Error Sampling selects the query based on the absolute difference between the estimated class probabilities of the two most likely classes [6, 27]. Tong and Koller developed an algorithm for performing active learning with SVMs by taking advantage of the duality between parameter space and feature space [29]. The algorithm attempts to shrink the version space as much as possible at each query.

There have been some attempts to combine semi-supervised learning with active learning. McCallum and Nigam leveraged a large pool of unlabeled documents in two ways—using EM and via density-weighted pool-based sampling [19]. Muslea et al. developed a multi-view algorithm, Co-EMT, which combines semi-supervised and active learning [21]. However, to our knowledge, our active EM algorithm is the first attempt to apply semi-supervised or active learning ideas to recover from label noise in the training data.

CONCLUDING REMARKS

This paper addressed the problem of supervised learning with noisy labels. This problem arises in the TaskTracer system, which provides an "activity-aware" overlay on Microsoft Windows XP. TaskTracer combines explicit user declarations of the "current activity" with a learned classifier that attempts to detect cases where the user changes activities without declaring the new activity. Our fundamental hypothesis is that neither a system based solely on explicit user declarations nor a system based solely on unsupervised learning will be able to work as well as a system that combines user declarations with statistical learning.

We demonstrated that combining semi-supervised learning with active learning can significantly improve the precision. The users painstakingly provided correct labels via an annotation tool, which allowed us to evaluate the performance of our semi-supervised and active learning algorithms. The combination of active learning with semi-supervised EM, which we called Active EM, gave the best precision at most levels of coverage. It performed better than Active Learning alone, and it performed better than semi-supervised EM alone.

There are additional sources of information that our method does not exploit, but that could improve performance. First, the duration of the WDS could be useful for prediction, because short-duration WDSs are more likely to be mislabeled (e.g., because the user changes the declared activity and then closes several windows, which results in several short WDSs). Second, the time elapsed between the moment when the user declares the activity and the start of the WDS is also important. It is reasonable to assume that the user's declaration is most accurate during the few minutes immediately following the declaration, and that it becomes less accurate over time. In other work, we have fit duration models to the activity episodes, and these could be applied to predict when one episode is likely to have ended and the user has switched to a different activity. Finally, we are studying change-point detection methods based on analyzing a sliding window of WDSs, which will allow us to pool evidence from multiple WDSs in predicting when an activity switch has occurred.

ACKNOWLEDGMENTS

The authors would like to acknowledge the anonymous reviewers' insightful comments which were helpful, as well as the help of the TaskTracer team members. This project was supported in part by the NSF under grant IIS-0133994 and by the DARPA under grant no. HR0011-04-1-0005 and contract no. NBCHD030010.

REFERENCES

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

2. D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

3. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, pages 506–519, 2003.

4. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc.of COLT-98*, pages 92–100, 1998.

5. C. E. Brodley and M. Friedl. Identifying mislabeled training data. *JAIR*, 11:131–167, 1999.

6. D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

7. D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

8. A. Dempster, N. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, Series B 39:1–38, 1977.

9. M. Dredze, T. Lau, and N. Kushmerick. Automatically classifying emails into activities. In *Proc. of IUI-06*, pages 70 – 77, 2006.

10. Y. Freund, H. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.

11. 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 *UAI-98*, pages 256–265, 1998.

12. E. Horvitz, A. Jacobs, and D. Hovel. Attention-sensitive alerting. In *UAI-99*, pages 305–313, 1999.

13. E. Horvitz, A. Jacobs, and D. Hovel. Learning and reasoning about interruption. In *Proc. of ICMI-03*, pages 20–27, 2003.

14. T. Joachims. Transductive inference for text classification using support vector machines. In *Proc. of ICML-99*, pages 200–209, 1999.

15. V. Kaptelinin. UMEA: translating interaction histories into project contexts. In *SIGCHI*, pages 353–360, 2003.

16. M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, pages 983–1006, 1998.

17. N. Kushmerick and T. Lau. Automated email activity management: an unsupervised learning approach. In *Proc. of IUI-05*, pages 67 – 74, 2005.

18. D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proc. of SIGIR-94*, pages 3–12, 1994.

19. A. McCallum and K. Nigam. Employing EM in pool-based active learning for text classification. In *Proc.of ICML-98*, pages 350–358, 1998.

20. T. M. Mitchell, S. H. Wang, Y. Huang, and A. Cheyer. Extracting knowledge about users' activities from raw workstation contents. In *Proc.of AAAI-06*, 2006.

21. I. Muslea, S. Minton, and C. Knoblock. Active + semi-supervised learning = robust multi-view learning. In *Proc.of ICML-02*, pages 435–442, 2002.

22. K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. of CIKM-00*, pages 86–93, 2000.

23. K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

24. 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.

25. D. Pierce and C. Cardie. Limitations of co-training for natural language learning from large datasets. In *Proc. of EMNLP*, pages 1–9, 2001.

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

27. M. Saar-Tsechansky and F. Provost. Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178, 1994.

28. J. Shen, L. Li, T. Dietterich, and J. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *Proc. of IUI-06*, pages 86–92, 2006.

29. S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proc. of ICML-00*, pages 999–1006, 2000.