Real-Time Detection of Task Switches of Desktop Users

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

Abstract

Desktop users commonly work on multiple tasks. The TaskTracer system provides a convenient, low-cost way for such users to define a hierarchy of tasks and to associate resources with those tasks. With this information, TaskTracer then supports the multi-tasking user by configuring the computer for the current task. To do this, it must detect when the user switches the task and identify the user's current task at all times. This problem of "task switch detection" is a special case of the general problem of change-point detection. It involves monitoring the behavior of the user and predicting in real time when the user moves from one task to another. We present a framework that analyzes a sequence of observations to detect task switches. First, a classifier is trained discriminatively to predict the current task based only on features extracted from the window in focus. Second, multiple single-window predictions (specifically, the class probability estimates) are combined to obtain more reliable predictions. This paper studies three such combination methods: (a) simple voting, (b) a likelihood ratio test that assesses the variability of the task probabilities over the sequence of windows, and (c) application of the Viterbi algorithm under an assumed task transition cost model. Experimental results show that all three methods improve over the single-window predictions and that the Viterbi approach gives the best results.

1 Introduction

Today's desktop information workers continually shift between different tasks (i.e., projects, working spheres). For example, Gonzalez and Mark [2004], in their study of information workers at an investment firm, found that the average duration of an episode devoted to a task was slightly more than 12 minutes, and that workers typically worked on 10 different tasks in a day. Furthermore, the information needed for each task tends to be fragmented across multiple application programs (email, calendar, file system, file server, web pages, etc.). The TaskTracer system [Dragunov *et al.*, 2005] is one of several efforts that seek to address this problem by creating "task-aware" user interfaces for the information worker.

TaskTracer provides a convenient way for the user to define a hierarchy of tasks and associate information resources (email messages, files, folders, web pages) with those tasks. The user can declare a "current task" (via a Task Selector component in the window title bar), and TaskTracer then configures the desktop in several ways to support the selected task. For example, it supplies an application called the Task Explorer, which presents a unified view of all of the resources associated with the current task and makes it easy to open those resources in the appropriate application. It also predicts which folder the user is most likely to access and modifies the Open/Save dialog box to use that folder as the default folder. If desired, it can restore the desktop to the state it was in when the user last worked on the task, and so on.

This approach of requiring the user to declare the current task 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, instant message). 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.

To address this problem, we wish to apply machine learning methods to automatically detect task switches. When a task switch is detected, we would like TaskTracer to pop up a "balloon alert" asking the user for permission to change the current declared task. To be usable, this task switch detector must be highly precise (i.e., have very low false alarm rate), and it must be timely (i.e., it must detect the task switch as soon as possible after it occurs so as to avoid interrupting the user once he or she is fully engaged in the new task). Finally, it must consume minimal CPU resources so that it does not interfere with the work the user is trying to accomplish. We call this task switch detector the TaskPredictor.

The first TaskPredictor for TaskTracer was developed by Shen et al.[2006] and deployed in TaskTracer in 2006. This TaskPredictor is a classifier trained to predict the user's current task based only on features describing the window currently in focus. In the remainder of this paper, we will refer to this as the Single Window Classifier (SWC). Although the SWC achieves fairly high accuracy, it is unusable in practice because it produces far too many task-switch false alarms.

This paper describes a set of experiments to develop an improved TaskPredictor. Our basic idea is to develop algorithms that analyze the sequence of predictions from the SWC to make more reliable task switch predictions.

The remainder of this paper is structured as follows. First, we describe the Single Window Classifier in detail. Then we present a framework that we have applied to the task switch detection problem. This framework makes the task switch decision based on a metric that combines multiple task predictions over time. Three different metric functions are proposed in this paper. Third, we show experimental results on both synthetic and real user data. We conclude the paper with a review of related work and a discussion of future work.

2 TaskTracer and the Single Window Classifier (SWC)

TaskTracer operates in the Microsoft Windows environment and collects a wide range of events describing the user's computer-visible behavior. 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 the raw event stream, the SWC extracts a sequence of *Window-Document Segments* (WDSs). A WDS consists of a maximal contiguous segment of time in which a window is in focus and the name of the document in that window does not change. The SWC extracts information about the window title and the file pathname from each WDS. Then it heuristically segments these into a bag of "words" and creates a binary variable x_j in the feature set for each unique word. To put more weight on long-duration WDSs, TaskPredictor creates multiple training instances for each WDS. For a WDS that lasts t seconds, TaskPredictor generates $n = \lceil \frac{5}{1+5\exp(-0.1t)} \rceil$ identical training instances.

For the sake of both accuracy and speed, the information gain [Yang and Pedersen, 1997] of each feature is computed and then the 200 features with the highest (individual) predictive power are retained and the rest are discarded. We will refer to these 200 features as the "informative" features .

It is reasonable to ask whether feature selection causes a large number of WDSs to have empty bags of words (i.e., to be uninformative). We checked this on one user's 2-month TaskTracer dataset. This dataset contains 781 distinct words. We discretized time into 60-second periods and applied feature selection to the entire dataset. Figure 1 shows the fraction of episodes that have at least one informative feature as a function of the number of features selected. With 200 features, more than 95% episodes still have informative features.

The first version of our TaskPredictor worked by applying the SWC to predict the task of every informative WDS. If the predicted task was different than the user's declared task and if the prediction was sufficiently confident, then TaskPredictor issued a task switch alarm. Initial experience with this TaskPredictor revealed that it was very sensitive to noise and that it issued many false alarms. Clearly, we need a more sophisticated technique to solve this problem.

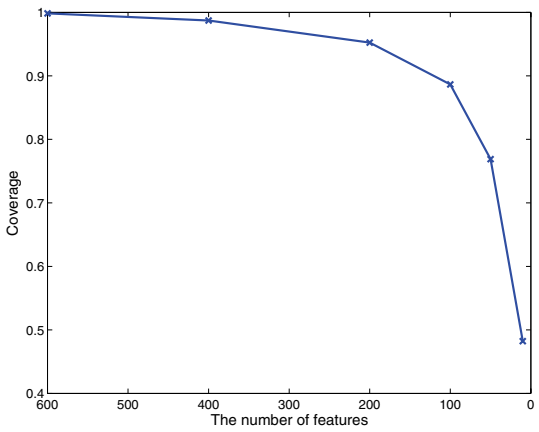

Figure 1: Impact of feature selection. Fraction of one-minute intervals that have informative observations as a function of the number of selected features.

3 Switch Detection With Informative Observations

Detecting switches from individual task predictions fails mainly because decisions based on just one observation are not reliable: given the observations $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$, the SWC only uses $\mathbf{x}_t$ to predict the task at time t. Instead, we should detect switches based on the previous ℓ observations $\{\mathbf{x}_{t+1-\ell}, \ldots, \mathbf{x}_t\}$. The larger value of ℓ can bring more information to make the task switch predictions. However, this comes at the cost of reducing the timeliness of the predictions, because roughly we have to wait for $\ell/2$ observations before the system has enough information to reliably detect the task switch.

3.1 A Framework to Detect Task Switches

Figure 2 presents our framework for detecting task switches. We execute the detector every f seconds or when an event is received. If the bag of features for a WDS contains no informative features (i.e., it is the null bag), then it is ignored. This usually happens for short-duration pop-up windows, empty Internet Explorer windows, new office documents ("Document1"), and so on.

The informative WDS observations are pushed into a first-in-first-out queue $\mathbf{X}$ of maximum length ℓ. Once $\mathbf{X}$ contains ℓ elements, each new element added to $\mathbf{X}$ causes the oldest element to be removed. The function SWITCHMETRIC computes the switch metric value Λ from $\mathbf{X}$ and also returns the predicted new task as discussed below. If Λ exceeds a threshold, a task-switch alert is shown to the user, and the user is asked to confirm it. If the user agrees, then the current task is changed to the new task, otherwise it is left unchanged.

The variable SILENTNUMBER stores the number of informative observations since the last switch alarm was made. To avoid annoying the user, the method does not raise any new alarms for at least the next r informative observations after a switch alarm.

If no event occurs for a long time (more than d seconds), then there is no point in predicting a switch, so we can avoid

Procedure SWITCHDETECTOR(ℓ, d, r, threshold)

Input: ℓ – we will store ℓ observations in queue $\mathbf{X}$
d – efficiency delay; skip switch computation if WDS has lasted longer than d seconds
r – recovery time; keep silent for the next r informative observations after a switch alarm
NOEVENTTIME $\leftarrow 0$
SILENTNUMBER $\leftarrow \infty$
do (every f seconds or if an event happens)
if (no event happened)
NOEVENTTIME $\leftarrow$ NOEVENTTIME $+ f$
else NOEVENTTIME $\leftarrow 0$
//skip inference if nothing has happened for d seconds
if (NOEVENTTIME $> d$) **continue**
//update the observation queue $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$
$\mathbf{x}_{now} \leftarrow$ GETOBSERVATION()
if ($\mathbf{x}_{now}$ is not null)
UPDATEQUEUE($\mathbf{X}, \mathbf{x}_{now}, \ell$)
SILENTNUMBER $\leftarrow$ SILENTNUMBER $+ 1$
else continue
if (SILENTNUMBER $\leq r$) **continue**
(Λ, PREDICTEDTASK) $\leftarrow$ SWITCHMETRIC($\mathbf{X}$)
if ($\Lambda >$ threshold)
SILENTNUMBER $\leftarrow 0$
if (POPUPALERT(PREDICTEDTASK) == OK)
SETCURRENTTASK(PREDICTEDTASK)
until the system is shutdown

Figure 2: Generic framework for detecting task switches.

the cost of computing the switch metric. We do not want $\mathbf{X}$ to be full of identical observations from the same long-duration WDS. Hence, there is no need to update $\mathbf{X}$ either.

With some user studies, we set the default parameters for this framework as follows: f = 5, d = 25, and r = 10.

3.2 Metric Functions

We now describe three methods for computing metric functions based on the FIFO queue of informative observations $\mathbf{X}$. First we describe the baseline method, which is similar to what has been deployed in the latest TaskTracer release.

Baseline Method

The Baseline Method applies the SWC to each observation individually. Given an observation $\mathbf{x}$, the SWC returns three things: a predicted class label $\hat{y}$, a probability distribution $\Theta = P(y|\mathbf{x})$ over the possible class labels, and a confidence measure that estimates $P(\mathbf{x})$. The baseline method considers a prediction to be confident if $P(\mathbf{x})$ exceeds a given threshold [Shen *et al.*, 2006]. It compares the class labels of the two most recent confident predictions and declares a task switch alarm if they are different.

Class Probability Voting

A simple solution to combine multiple predictions is voting. Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ be the queue of informative observations. The basic idea is to vote for the task of $(\mathbf{x}_1, \ldots, \mathbf{x}_{\ell/2})$ and the task of $(\mathbf{x}_{\ell/2+1}, \ldots, \mathbf{x}_\ell)$ and see whether they are the same. First, compute $y_a = \arg\max_y \sum_{i=1}^{\ell/2} P(y|\mathbf{x}_i)$ and $y_b = \arg\max_y \sum_{i=\ell/2+1}^{\ell} P(y|\mathbf{x}_i)$. Then calculate the switch metric as

$$\Lambda = I[y_a \neq y_b] \cdot \left\{ \sum_{i=l}^{\ell/2} P(y_a|\mathbf{x}_i) + \sum_{i=\ell/2+1}^{\ell} P(y_b|\mathbf{x}_i) \right\}, \quad (1)$$

where $I[p] = 1$ if p is true and -1 otherwise. Throughout this paper, we train support vector machines to do class probability estimation [Platt, 1999; Wu *et al.*, 2004]. This voting method votes the class probability distributions to calculate the metric. It is straightforward and cheap, but it doesn't take into account the variability of the task probabilities.

Likelihood Ratio Scores

Instead of just looking at the sum of the predicted class probability distributions, we can also consider the variability of those distributions. Although voting may suggest a task switch, if the class probability distributions within each half of $\mathbf{X}$ are highly variable, this suggests low confidence in the prediction.

Let Θ_i be the posterior probability distribution over the tasks given observation $\mathbf{x}_i$, and μ_a, μ_b be the true means of $\{\Theta_1, \ldots, \Theta_{\ell/2}\}$ and $\{\Theta_{\ell/2+1}, \ldots, \Theta_\ell\}$ respectively. We seek to test the hypothesis $H_0 : \mu_a = \mu_b$ against the alternative $H_1 : \mu_a \neq \mu_b$. Developed by Neyman and Pearson, the *likelihood ratio test* provides a general methodology to test a two-sided alternative against a null hypothesis [Mukhopadhyay, 2000]. The central limit theorem states that the sample mean is approximately a Gaussian distribution given a sufficiently large sample. Thus, we can compute the switch metric Λ as the Gaussian likelihood ratio score

$$\Lambda = (\overline{\Theta}_a - \overline{\Theta}_b)^T \left(\frac{2}{\ell}\Sigma_a + \frac{2}{\ell}\Sigma_b \right)^{-1} (\overline{\Theta}_a - \overline{\Theta}_b), \quad (2)$$

where $\overline{\Theta}_a$ and $\overline{\Theta}_b$ are the sample means and Σ_a and Σ_b are the sample covariance matrices of $\{\Theta_1, \ldots, \Theta_{\ell/2}\}$ and $\{\Theta_{\ell/2+1}, \ldots, \Theta_\ell\}$ respectively. We should reject H_0 for large values of Λ. If we assume $\forall i \neq j$, y_i and y_j are independent, and treat Σ_a and Σ_b as diagonal matrices, then the computation time is $O(\ell \cdot |y|)$ excluding the expense for the SWC to compute $P(y_j|\mathbf{x}_i)$.

The central limit theorem suggests that ℓ should be quite large (e.g., > 30). However, very large ℓ values will substantially delay the task switch alarms. Fortunately, our experimental results are good even when $\ell = 8$.

Minimal Transition Costs

Our third metric is based on hypothesizing a task switch cost and applying an additive conditional cost formula similar to that of hidden Markov models (HMMs). We define the cost of labeling $\mathbf{X}$ by $\mathbf{Y}$ as

$$C(\mathbf{Y}|\mathbf{X}) = \sum_{1 \leq i \leq \ell} C_a(y_i|\mathbf{x}_i) + \sum_{1 < i \leq \ell} C_t(\mathbf{x}_i|\mathbf{x}_{i-1}), \quad (3)$$

where C_a is the *atemporal cost* of labeling observation i with y_i and C_t is the *transition cost* of moving from task y_{i-1} to task y_i. We define C_a as the negative log likelihood $C_a = -\log P(y_i|\mathbf{x}_i)$. As in Fern et al. [2005], we define the transition cost function as $C_t = c \cdot \delta[y_i \neq y_{i-1}]$, where c is a real and $\delta[p] = 1$ if p is true and 0 otherwise. This model simply charges an equal cost for all task switches, even though it's possible that some switches are more likely than others.

We assume that the sequence $\mathbf{X}$ is short enough that it only contains 0 or 1 switches. We are concerned with whether there is *one* switch for sequence $\mathbf{X}$. So we compare the minimal cost under the assumption of 0 switches to the minimal cost under the assumption of one switch:

$$\alpha = \min_a C(y_1 = y_2 = \ldots = y_\ell = a|\mathbf{X}), \quad (4)$$

$$\beta = \min_{i,a\neq b} C(y_1 = \ldots = y_i = a, y_{i+1} = \ldots = y_\ell = b|\mathbf{X}). \quad (5)$$

We apply the Viterbi algorithm to find these minimal costs. Given $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$, let α_{ij} be the minimal cost if we predict the task at time i as y_j and predict no switch until time i. Let β_{ij} be the minimal cost if we predict the task at time i as y_j and that one switch has occurred prior to time i. For each y_j, we initialize $\alpha_{1j} = C_a(y_j|\mathbf{x}_1)$ and $\beta_{1j} = +\infty$. For task y_j, if we assume that no switch has occurred until time i, then α_{ij} is simply the sum of the atemporal costs plus $\alpha_{(i-1)j}$. If we assume that one switch has occurred prior to time i, there are two possibilities: this switch occurred at time i or before time i. We then pick the choice with the smaller cost and add the atemporal cost to give β_{ij}. Thus, $\forall 1 < i \leq \ell$, we can recursively calculate the cost for each y_j at time i as

$$\alpha_{ij} = \alpha_{(i-1)j} + C_a(y_j|\mathbf{x}_i), \quad (6)$$

$$\beta_{ij} = \min\{\beta_{(i-1)j}, \min_{k\neq j} \alpha_{(i-1)k} + c\} + C_a(y_j|\mathbf{x}_i). \quad (7)$$

The minimal costs over the entire sequence $\mathbf{X}$ will be $\alpha = \min_j \alpha_{\ell j}$ and $\beta = \min_j \beta_{\ell j}$. We set the value of the metric as their difference:

$$\Lambda = \alpha - \beta. \quad (8)$$

This Viterbi search will take $O(\ell \cdot |y|^2)$ time excluding the expense for $C_a(y_j|\mathbf{x}_i)$. In step i we only need to calculate $C_a(y_j|\mathbf{x}_i)$ for each y_j, and these values can be cached for future reuse.

4 Experimental Results

We tested the framework and the three task switch metrics on synthetic data and on data from four real users.[1]

We generated 20 synthetic datasets using the following process. Each dataset contained 8 tasks and 500 distinct words. For each task, its multinomial distribution over words was randomly generated so that 150 of the words were approximately 7 times more likely to appear than the remaining 350 words. Then, we sequentially generated 300 episodes: first we chose a task y uniformly at random and generated the length of the episode m uniformly in the range $(10 \leq m \leq 60)$. Then we generated m observations for task y according to its multinomial distribution such that each observation lasts 1 second and contains less than 5 words.

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

Table 1: Datasets for Evaluating Switch Detection (# of words is computed after stoplist and stemming)

Data Set	FA	FB	FC	SA
size(in months)	6	2	4	2.5
# of switches	374	246	1209	286
# of tasks	83	51	176	8
# of words	575	781	1543	514

The real user data was obtained by deploying TaskTracer on Windows machines in our research group and collecting data from four users, whom we referred to as FA, FB, FC, and SA. The collected data is summarized in Table 1. Each episode in these data sets was hand-labeled with the associated task, although these labels probably exhibit considerable levels of noise.

We now present the results for two performance metrics: COAP and precision.

COAP given the size of the queue. We measure the probability that segment boundaries are correctly identified by the *co-occurrence agreement probability* (COAP) [Beeferman *et al.*, 1999; McCallum *et al.*, 2000]. The sequence of observations can be divided into segments using either the observed or the predicted task switch points. Consider two arbitrary time points t_1 and t_2 in the segmented sequence. Either t_1 and t_2 belong to the same segment, or they belong to different segments. The COAP is computed by choosing a probability distribution D over (t_1, t_2) and measuring the probability that the observed and the predicted segmentations agree on whether these two points belong to the same or different segments. In our case, D is the uniform distribution over all pairs (t_1, t_2) such that $|t_1 - t_2| \leq 30$ time units. For the synthetic data set, the time unit was 1 second; for the real user data, the time unit was 60 seconds.

For each dataset, the data is ordered according to time and divided into three equal time intervals: A, B, and C. We first

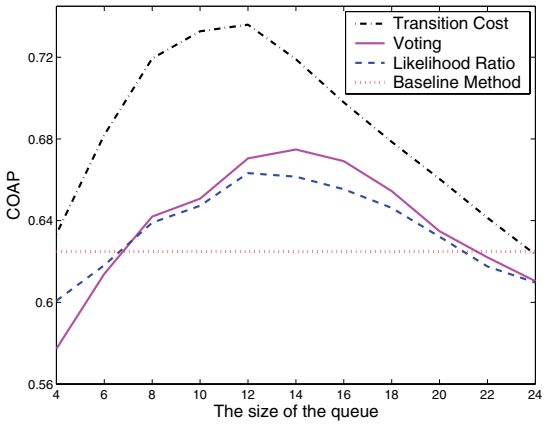

Figure 3: The average COAP values of the synthetic data as a function of ℓ, the length of the observation queue $\mathbf{X}$

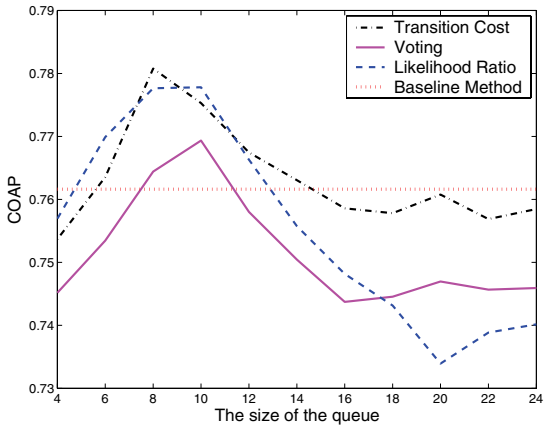

Figure 4: The average COAP values of the real user data as a function of ℓ

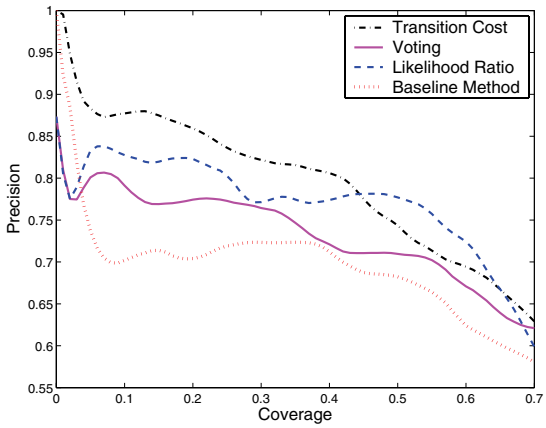

Figure 5: The average precision values as a function of the coverage for real users given 10 observations, created by varying the prediction threshold.

train the Single Window Classifier using A. Then for a given switch detection algorithm, we vary the threshold and evaluate on B to choose the best threshold. Recall that we issue a switch alarm when the metric value is larger than the threshold. Finally, we retrain the SWC using A+B and apply the learned SWC and threshold value to evaluate on C. The results on C are plotted in Figure 3 and 4.

From the plots, ℓ (the size of informative observation queue) is crucial to the accuracy of methods. When ℓ is increased, the COAPs first increase and then decrease for all three metric functions. The initial increase is presumably due to the increase in the amount of information available. The subsequent decrease is presumably due to the fact that when the queue $\mathbf{X}$ is too long, it contains more than one task switch, but our framework assumes only 0 or 1 task switches occur. There might be 2 or even more switches when ℓ is large. The maximal COAPs are reached when ℓ is around 12 for the synthetic data and around 10 for the real user data. The highest COAP value of any metric function is larger than that of the baseline method.

The minimal transition cost metric gives the best performance, particularly on the synthetic data. One reason may be that unlike the other metrics, it is not constrained to predict the switch point in the center of the observation queue $\mathbf{X}$. The likelihood ratio metric is more conservative in predicting switches, since it will not issue any alarm if the predicted $P(y|\mathbf{x}_i)$ is not stable. Thus it misses some real switches, but it makes fewer false alarms. This sometimes leads to a low COAP value. The voting method is the least effective.

Precision given coverage. We measured precision and coverage as follows. If a correct (user-labeled) switch occurs at time t and a switch is predicted to occur between $t-60$ and $t+180$, then the first such switch prediction is considered to be correct. However, subsequent predictions in the same time interval are counted as errors. Precision is the probability that a switch prediction is correct. Coverage is the probability that a user-labeled switch is predicted correctly.

We adopted an on-line learning methodology to evaluate the algorithms on the real user data. We divided the data into days and assign d_0 to be the day that is 25% of the way through the data set. This defines an initialization period. For each day d from d_0+1 to the end of the data set, we trained the system on the data from days 0 through $d-1$ and then evaluated its performance on day d. The precision given coverage for $\ell = 10$ informative observations is plotted in Figure 5.

Our framework with any of the metric functions outperforms the baseline method. These results show that using more observations does reduce the false alarms. The minimal transition cost metric gives the best results for coverage below 40%. Compared to the baseline method, the minimal transition cost metric can improve precision by more than 15%. The likelihood ratio metric gives the best results for coverage from 40% to 65%. Voting is the worst of the three metrics.

5 Related Work

Our method of task switch detection is based on task recognition and prediction, for which many methods have been developed (see Shen et al., [2006] for a review). There are also some attempts in understanding the user's interest in information retrieval area [Ohsawa and Yachida, 1997].

Task switch detection is also related to change-point detection [Chen and Gopalakrishnan, 1998; Guralnik and Srivastava, 1999]. The standard approach of change-point detection has been to (a) apriori determine the number of change-points that are to be discovered, and (b) decide the function that will be used for curve fitting in the interval between successive change-points [Guralnik and Srivastava, 1999]. These approaches usually do not employ *supervised* learning techniques. Our switch detection method sheds some light on such problems: we can hand-label observations and train a switch detector to predict change points.

Task switch detection is different from novelty detection [Schölkopf *et al.*, 2000; Campbell and Bennett, 2001; Ma and Perkins, 2003] and first story detection [Allan *et al.*, 2000]. Novelty detection, or anomaly detection, tries to automatically identify novel or abnormal events embedded in a large body of normal data. One application is to liberate scientists from exhaustive examination of data by drawing their

attention only to unusual and "interesting" phenomena [Ma and Perkins, 2003]. The problem is typically solved by learning a pattern to cover most normal data points. Any point outside of this pattern will be considered as a novelty. However, in switch detection, when the user is changing tasks, the current observation may be neither novel nor abnormal. First Story Detection (FSD) is the task of online identification of the earliest report for each news story as early as possible. Existing FSD systems usually compare a new document to all the documents in the past, and they predict a first story if the similarity score is smaller than some threshold. This method does not work for switch detection, because task switches often involve repeating activities that the user did in the past.

Text segmentation tries to predict where boundaries occur in text [Beeferman *et al.*, 1999]. Maximum entropy Markov models [McCallum *et al.*, 2000] and conditional random fields [Lafferty *et al.*, 2001] have been successful in natural language tasks. With some additional requirements, it is possible to apply these state-of-the-art models to detect task switches. First, since task switch detection is done in real time, we should do efficient filtering (instead of smoothing). Second, the cost for wrong predictions is unbalanced: A false alarm costs a lot; it is more acceptable to miss some switches.

6 Conclusions And Further Work

In this paper, we introduced a special case of the general problem of change-point detection, which we term task switch detection. This involves monitoring the user's behavior and predicting when the user switches to a different task. To reduce false alarms, we made our decision based on multiple informative observations–observations exhibiting one or more of the 200 most informative features. We tested this framework in the TaskTracer system and compared three switch metrics, each computed from information produced by a learned Single Window Classifier (SWC). We found that the metric based on an assumed task switch transition cost and the Viterbi algorithm gave the best results.

We are exploring four directions for future improvement. First, we are testing methods for overcoming noise in the user's task switch labels by treating those labels as noisy measurements of the true labels and training the SWC using EM. Second, we are studying methods of active learning to reduce the number of labels that the user needs to provide. Third, we want to learn episode-duration models and apply them (e.g., in place of the fixed recovery time parameter r). Finally, we would like to replace our simple decision threshold with a decision-theoretic procedure based on the estimated costs and benefits of interrupting the user with a task switch alarm.

Acknowledgments

Supported by the NSF under grant IIS-0133994 and by the DARPA under grant no. HR0011-04-1-0005 and contract no. NBCHD030010. The authors thank the TaskTracer team.

References

[Allan *et al.*, 2000] J. Allan, V. Lavrenko, and H. Jin. First story detection in tdt is hard. In *Proc. of CIKM-00*, pages 374–381, 2000.

[Beeferman *et al.*, 1999] D. Beeferman, A. Berger, and J. Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1-3):177–210, 1999.

[Campbell and Bennett, 2001] C. Campbell and K. Bennett. A linear programming approach to novelty detection. In *NIPS 13*. 2001.

[Chen and Gopalakrishnan, 1998] S. Chen and P. Gopalakrishnan. Speaker, environment and channel change detection and clustering via the bayesian information criterion. In *DARPA speech recognition workshop*, 1998.

[Dragunov *et al.*, 2005] A. Dragunov, T. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proc. of IUI-05*, pages 75–82, 2005.

[Fern, 2005] A. Fern. A simple-transition model for relational sequences. In *IJCAI-05*, pages 696–701, 2005.

[Gonzalez and Mark, 2004] V. Gonzalez and G. Mark. "Constant, constant, multi-tasking craziness": Managing multiple working spheres. In *CHI 2004*, 2004.

[Guralnik and Srivastava, 1999] V. Guralnik and J. Srivastava. Event detection from time series data. In *Proc. of KDD-99*, pages 33–42, 1999.

[Lafferty *et al.*, 2001] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML-01*, pages 282–289, 2001.

[Ma and Perkins, 2003] J. Ma and S. Perkins. Online novelty detection on temporal sequences. In *Proc. of SIGKDD-03*, pages 613–618, 2003.

[McCallum *et al.*, 2000] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proc. of ICML-00*, pages 591–598, 2000.

[Mukhopadhyay, 2000] N. Mukhopadhyay. *Probability and Statistical Inference*. Marcel Dekker Inc., 2000.

[Ohsawa and Yachida, 1997] Y. Ohsawa and M. Yachida. An index navigator for understanding and expressing users coherent interest. *IJCAI-97*, 1:722 – 728, 1997.

[Platt, 1999] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. 1999.

[Schölkopf *et al.*, 2000] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *NIPS 11*. 2000.

[Shen *et al.*, 2006] 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.

[Wu *et al.*, 2004] T. Wu, C. Lin, and R. Weng. Probability estimates for multi-class classification by pairwise coupling. In *NIPS 16*. 2004.

[Yang and Pedersen, 1997] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of ICML-97*, pages 412–420, 1997.