AN ABSTRACT OF THE DISSERTATION OF

Jianqiang Shen for the degree of Doctor of Philosophy in Computer Science presented on March 19, 2009.

Title: Activity Recognition in Desktop Environments

Abstract approved: ______________________________

Thomas G. Dietterich

Knowledge workers are struggling in the information flood. There is a growing interest in intelligent desktop environments that help knowledge workers organize their daily life. Intelligent desktop environments allow the desktop user to define a set of "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. TaskTracer [Dragunov *et al.*, 2005] is one of these efforts. We classify activities into three levels: task as in TaskTracer, workflow, and operation. In this thesis, we concentrate on the two higher levels of activity – "task" and "workflow" – and we present an activity recognition solution for each of them. To recognize tasks, we employ various supervised learning algorithms. The first approach is based on batch training, which has a heavy requirement for CPU time and memory. It is also insensitive to user feedback, and it is likely to make repeated mistakes. To address these issues, we propose a novel online learning algorithm that is able to incorporate a richer set of features than our previous predictors. We prove an error bound for the algorithm and present experimental results that show improved accuracy and a 180-fold speedup on real user data. To recognize workflows, we first mine frequent workflow models from the TaskTracer event log. Discovering desktop workflows is difficult because they unfold over extended periods of time (days or weeks) and they are interleaved with many other workflows because of user multi-tasking. We describe an approach to discovering desktop workflows based on rich instrumentation of information flow actions such as copy/paste, SaveAs, file copy, attach file to email message, and save attachment. These actions allow us to construct a graph whose nodes are files, email messages, and web pages and whose edges are these information flow actions. A class of workflows that we call work procedures can be discovered by applying graph mining algorithms to find frequent subgraphs. After finding the workflow models, we create a Logical Hidden Markov Model (LHMM) for each mined model to represent the workflow in a form suitable for real-time recognition using the WARP relational inference algorithm [Natarajan *et al.*, 2008].

# Activity Recognition in Desktop Environments

by

Jianqiang Shen

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented March 19, 2009
Commencement June 2009

# ACKNOWLEDGEMENTS

I am profoundly grateful to my advisor, Tom Dietterich. I was very fortunate to have Dr. Dietterich as my mentor. I have benefited greatly from Dr. Dietterich's high standards on quality and elegance of scientific work. His tireless pursuit of excellence in research, teaching, advising, and every other aspect of his academic work is truly inspirational. He has been incredibly supportive and patient with me during the past 5 years.

I would like to thank my committee members, Alan Fern, Jon Herlocker and Prasad Tadepalli for their excellent suggestions and thought-provoking questions. I have learned a great deal from their work and their influence on this thesis is immense. In particular, Dr. Fern's research on sequential inference and intelligent virtual assistants have inspired my work. Dr. Herlocker is always full of enthusiasm and good thoughts. He is one of the parents of TaskTracer, and his ideas have greatly shaped this thesis. I am grateful to Dr. Tadepalli for teaching me the beauty of AI and kindly answering my questions. I thank Dr. Rod Harter for taking his time off to serve in my committee.

I couldn't have made it this far without my wonderful collaborators of the TaskTracer project. Jed Irvine, Kevin Johnsrude, Simone Stumpf and Michael Slater have made significant contributions to the development of TaskTracer. Lida Li helped develop TaskPredictor, and I have been inspired by our research discussion. I'm also indebted to other team members. They include but are not limited to Xinlong Bao, Chen Cao, Fredric Carl, Phuoc Do, Erin Fitzhenry, Michael Goodman, Brenton Kirschner, Stephen Kolibaba, Twinkle Lettkeman, Ben Porter, Anh Tran, Mark Vulfson. I thank Dr. Hung Bui of SRI International for providing me with the WARP software package and inspirational research discussions. It has been a great pleasure working with those smart people.

Werner Geyer and Michael Muller hosted me at IBM Research in the summer of 2006. Their wide-ranging research on social software and collaborative environments enlightened much of my work. I am also grateful to them for having faith in my ideas and opening the doors of their lab to me.

I am thankful to Weng-Keen Wong for leading me to the area of Bayesian learning. I have benefited in many aspects from our conversations.

Finally, I would like to thank my family. They have provided tremendous support during my graduate study and tolerated my occasional negligence during intense periods. This thesis is dedicated to them.

# TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

# LIST OF FIGURES

LIST OF FIGURES (Continued)

# LIST OF TABLES

# LIST OF APPENDIX FIGURES

## DEDICATION

To those who love me and are loved by me.

# Chapter 1 – Introduction

Knowledge workers, such as professors, managers, and engineers, perform many different activities in their daily work life. An intelligent user interface helps 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. Such intelligent software usually possesses one or more of the following abilities in order to assist the user: *organizing* the information, *adapting* the interface and *understanding* the user's activity.

Most operating systems (e.g., Windows, Linux, Mac OS) let the user organize his/her files in a hierarchical way. While this is convenient when accomplishing a task that only involves files from the same source, it becomes more challenging when the task involves heterogeneous information (e.g., involving both email messages and doc files). Some intelligent software can organize information based on the user's explicitly-defined or implicitly-inferred activities [Bellotti *et al.*, 2003; Dragunov *et al.*, 2005; Rattenbury and Canny, 2007]. This allows the user to access the information related to a specific activity more easily and thus accomplish the task more efficiently.

Instead of working in a one-size-fits-all way, some intelligent user interfaces try to personalize the desktop in order to enhance user productivity. They may re-configure the desktop to allow the access to some specific information easier

[Bao *et al.*, 2006], or they may proactively recommend the information related to a specific activity to the user [Shen *et al.*, 2008], or they may automatically generate interfaces that are tailored to a user's motor capabilities [Weld *et al.*, 2003].

Understanding what the user is trying to accomplish can supply the grounds to provide some appropriate assistance. Some systems try to generalize a repeatable procedure from the user's demonstration [Lau *et al.*, 2003; Lau *et al.*, 2004]. They then execute such procedures automatically or semi-automatically to improve work efficiency. Some systems observe a sequence of events and try to understand the goal of the user in order to respond to it [Andrews *et al.*, 2004; Horvitz *et al.*, 1998; Philipose *et al.*, 2003; Shen *et al.*, 2006].

TaskTracer [Dragunov *et al.*, 2005] is an intelligent activity management system which performs the above functions: it tries to organize the user's information based on activities, adapt the interface to make information access easier, and predict which activity the user is doing. TaskTracer improves the user's work efficiency by effectively helping the user organize and re-find information. For example, it supplies an application called the TaskExplorer, 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. TaskTracer also predicts which folder the user is most likely to access and modifies the Open/Save dialog box to minimize the expected number of mouse clicks.

To correctly organize the information and appropriately adapt the user interface, it is critical for TaskTracer to understand which activity the user is executing. This dissertation presents our effort to apply machine learning technology to activity recognition in the desktop environment. More specifically, we seek to accomplish the following goals: 1) determine if the user has switched tasks, 2) predict what is the current task, 3) estimate the state of workflow execution.

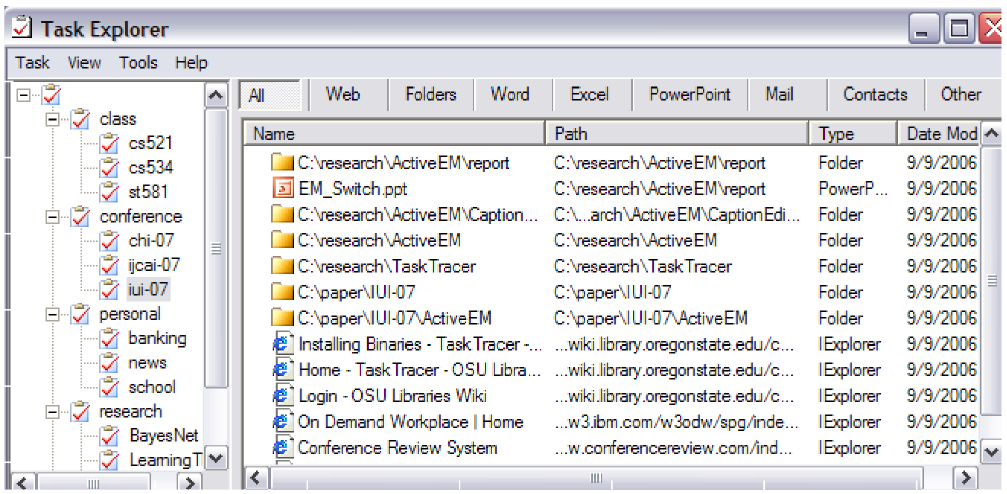
In the remainder of this chapter, we first introduce the TaskTracer system, which is the activity management system upon which our work is built. Next, we discuss related work. We then describe the activity recognition problem.

## 1.1 Software Aiming to Improve the Efficiency of Desktop Users

Knowledge workers are struggling in the information flood. Many intelligent assistants have been proposed to help knowledge workers organize their daily life. TaskTracer [Dragunov *et al.*, 2005; Shen *et al.*, 2006] is one such effort.

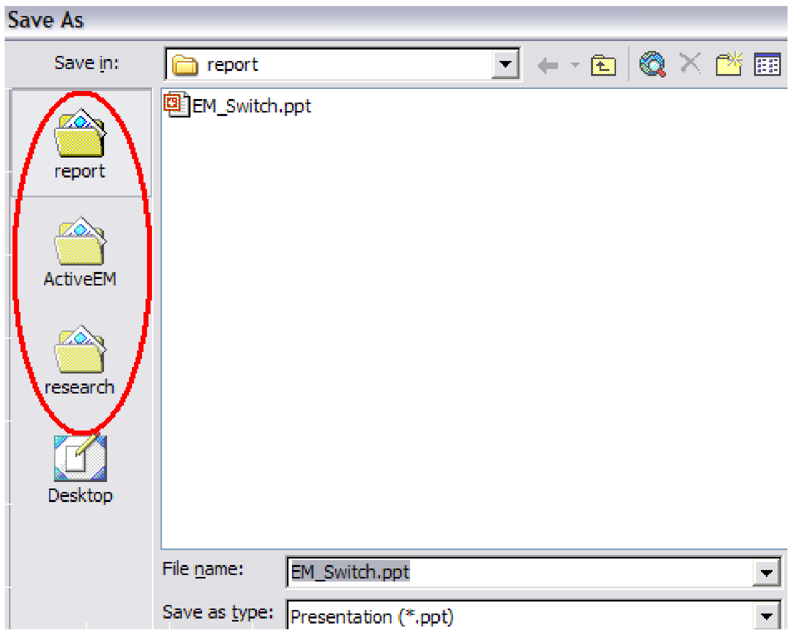
### 1.1.1 The TaskTracer System

The TaskTracer system is an intelligent activity management system [Kaptelinin, 2003; Quan and Karger, 2003; Bellotti *et al.*, 2003; Geyer *et al.*, 2006] that helps knowledge workers manage their work based on two assumptions: (a) the user's work can be organized as a set of ongoing tasks such as "Write TaskTracer Paper" or "Take CS534 Class", (b) each task is associated with a set of *resources*. "Resource" is an umbrella term for documents, folders, email messages, email contacts, web pages and so on. TaskTracer collects events generated by the user's computer-visible behavior, including events from MS Office, Internet Explorer,



| Name | Path | Type | Date Mod |
|---|---|---|---|
| C:\research\ActiveEM\report | C:\research\ActiveEM\report | Folder | 9/9/2006 |
| EM_Switch.ppt | C:\research\ActiveEM\report | PowerP... | 9/9/2006 |
| C:\research\ActiveEM\Caption... | C:\...arch\ActiveEM\CaptionEdi... | Folder | 9/9/2006 |
| C:\research\ActiveEM | C:\research\ActiveEM | Folder | 9/9/2006 |
| C:\research\TaskTracer | C:\research\TaskTracer | Folder | 9/9/2006 |
| C:\paper\IUI-07 | C:\paper\IUI-07 | Folder | 9/9/2006 |
| C:\paper\IUI-07\ActiveEM | C:\paper\IUI-07\ActiveEM | Folder | 9/9/2006 |
| Installing Binaries - TaskTracer -... | ...wiki.library.oregonstate.edu/c... | IExplorer | 9/9/2006 |
| Home - TaskTracer - OSU Libra... | ...wiki.library.oregonstate.edu/c... | IExplorer | 9/9/2006 |
| Login - OSU Libraries Wiki | ...wiki.library.oregonstate.edu/c... | IExplorer | 9/9/2006 |
| On Demand Workplace \| Home | ...w3.ibm.com/w3odw/spg/inde... | IExplorer | 9/9/2006 |
| Conference Review System | ...w.conferencereview.com/ind... | IExplorer | 9/9/2006 |

(a) Task Explorer



Save in: report

EM_Switch.ppt

File name: EM_Switch.ppt

Save as type: Presentation (*.ppt)

(b) Folder Predictor

Figure 1.1: The TaskExplorer (a) lists all resources related to the current task; the Folder Predictor (b) provides 3 predicted folders in the "places bar" at left

Windows Explorer, and the Windows XP operating system. The user can declare a "current task" (via a "Task Selector" UI), and TaskTracer records all resources as they are accessed and associates them with the current declared task. TaskTracer then configures the desktop in several ways to support the user:

- **Task Explorer**: This presents a unified view of all of the resources associated with the current task and makes it easy to launch those resources in the appropriate application (see Figure 1.1(a)). This supports interruption recovery by showing the user the most recent resources that they were working on for each task (as opposed to the global Recent Documents facility of Windows).

- **Folder Predictor**: This modifies the Open/Save dialogs so that they are task-aware (see Figure 1.1(b)). Folder Predictor predicts for all folders $f$ associated with the current task and computes the three folders that jointly minimize the expected number of clicks to reach $f$. The Open/Save dialog is initialized in the most likely of these folders, and shortcuts to all three folders are made available in the dialogue box [Bao *et al.*, 2006].

- **Time Reporting**: This allows the user to produce a report showing how much time was spent on each task during the most recent day, week, month, year, and so on.

- **Email Tagging**: Incoming and outgoing email messages are automatically tagged (via another machine learning component) with the task (or tasks) with which they are associated.

Our research is motivated by several TaskTracer use cases. First, to associate each accessed resource with the correct task, we would like TaskTracer to automatically detect task switches and correctly associate the resources with the corresponding task. Second, a significant portion of the desktop activity of knowledge workers involves executing workflows. It is challenging for the user to track the progress of the executing workflows. We are interested in detecting the initiation of a new workflow instance, tracking the progress of each workflow instance, and determining if the execution has been completed.

### 1.1.2 Related Work

Many intelligent systems have been proposed to help knowledge workers manage their work. Such intelligent software usually possesses one or more of the following abilities in order to appropriately assist the user: *organizing* the information, *adapting* the interface and *understanding* the user's activity.

**Organizing the information.** Activity management systems help knowledge workers manage their daily work life by organizing information based on activities. They are mainly based on two assumptions: (a) the desktop user switches between different activities, and (b) each activity is associated with a set of resources relevant to that activity. Those activities are usually some long-lived projects, such as "Writing PhD Dissertation", "Project TaskTracer" or "Teaching CS534". Such activities are called "tasks" in our TaskTracer system. Resources can be documents, photographs, podcasts, email messages, web pages, RSS feeds, social bookmarks, chats and so on. Activity management systems help with task switching and resource re-discovery by providing a context for organizing and accessing related resources.

Some systems focus on email by tracking tasks hidden in the user's inbox [Bellotti *et al.*, 2003; Dredze *et al.*, 2006; Kushmerick and Lau, 2005]. Activity-centric email clients, for example, enable users to associate incoming email messages with existing tasks. Other systems try to support heterogeneous resources besides email messages [Kaptelinin, 2003; Dragunov *et al.*, 2005]. These systems monitor the user's activities and track resources used when carrying out the task; they automatically organize and update these resources to make them easily available to the user when the task is resumed. Finally, there are activity management systems based on shared objects and collaboration around those objects [Quan and Karger, 2003; Geyer *et al.*, 2003; Moore *et al.*, ]. In contrast to the aforementioned single user systems, these systems provide integrated representations of diverse media that are assumed to be shared, such as email, IM, and web pages.

**Adapting the interface.** Adaptive systems enhance user productivity by automatically changing the interface in response to routine user behaviors. Traditional computer interfaces are one-size-fits-all. Users with little programming experience have very limited opportunities to customize an interface to their task and work habits [Weld *et al.*, 2003]. Adaptive systems automatically personalize the interfaces based on the user's motor ability, personal preference, or information requirements.

SUPPLE++ [Gajos *et al.*, 2007] models users' motor capabilities based on a one-time motor performance test and uses this model in an optimization process to generate a personalized interface. ARNAULD [Gajos and Weld, 2005] is a general interactive tool for eliciting user preferences concerning concrete outcomes and using this feedback to automatically learn a factored cost function. FolderPredictor [Bao *et al.*, 2006] is an application that applies a cost-sensitive prediction algorithm to the user's previous file access information and can reduce the cost of locating files in hierarchical folders. Malibu [Shen *et al.*, 2008] automatically changes the view of a side bar to support knowledge workers' activities.

**Understanding the user's activity.** With the ability to understand the user's activity, the intelligent assistant can provide more appropriate services to the user. There have been many attempts in machine learning to understand the user's goal or recognize the user's activity. To understand the user's goal, a programming by demonstration approach [Lau *et al.*, 2003; Lau *et al.*, 2004] is usually applied. The system generalizes a parameterized procedure from the user's repeated behavior. It then instantiates this learned procedure for a specific situation and provides automatic or semi-automatic operations.

To recognize the user's activity, researchers usually convert this inherently sequential problem into an ordinary supervised learning problem through the design of appropriate features [Andrews *et al.*, 2004]. Many of the applications are based on probabilistic models. The *Lumière* project centers on harnessing probability and utility to provide assistance to computer users [Horvitz *et al.*, 1998]. 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. [1999;

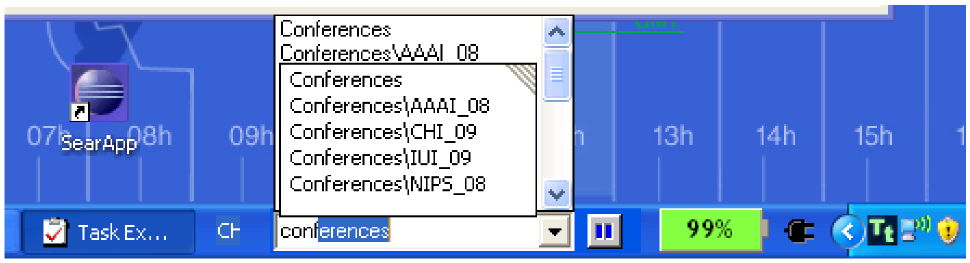


Figure 1.2: The screenshot of TaskSelector, an application that is located at the right bottom of the desktop and indicates the current task.

2003] 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) [Philipose *et al.*, 2003]. 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 that provide annotated object information. The model is very similar to a hidden Markov model.

Motivated by the requirements of the TaskTracer users, the focus of this dissertation will be trying to understand the user's activity.

## 1.2 Activity Recognition Problems

An intelligent user interface helps 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. Activity recognition (e.g., [Andrews *et al.*, 2004; Horvitz *et al.*, 1998; Horvitz *et al.*, 1999; Horvitz *et al.*, 2003; Philipose *et al.*, 2003]) plays an important role in such intelligent environments: the system observes a sequence of events, tries to understand the goal of the user, and attempts to respond to it.

The TaskTracer system [Dragunov *et al.*, 2005; Shen *et al.*, 2006] is one of the many efforts to design an intelligent user interface. It 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 windows tray area. See Figure 1.2), 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 reduce the user’s effort to locate files.

TaskTracer requires the user to explicitly declare the current task in order to correctly associate resources with tasks. This approach fails when the user is interrupted (e.g., by a phone call, instant message). The user typically changes documents, web pages, and so on without remembering to first inform TaskTracer. To address this problem, we designed and implemented some activity recognition components using machine learning methods.

A significant portion of the desktop activity of knowledge workers involves executing workflow procedures. These can be formally prescribed procedures (e.g., submitting requests for travel authorization) or informal, idiosyncratic procedures (e.g., keeping track of fantasy football results). We are also interested in design-

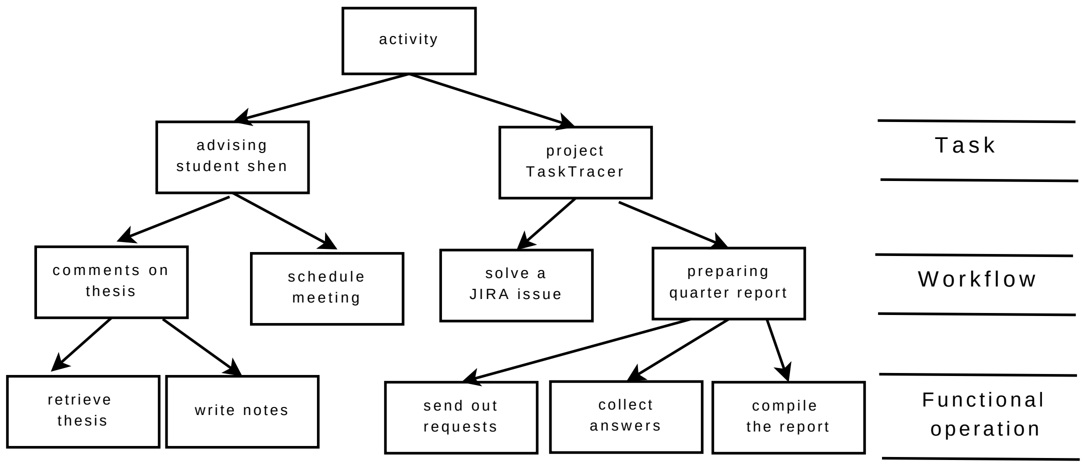


Figure 1.3: Example of an activity hierarchy. Each activity consists of several sub-activities.

ing an intelligent "To-Do" manager with the ability to recognize such workflow instances. We can then detect when a new workflow instance was initiated, track the current state of execution of each workflow instance, and determine when the execution was complete.

## 1.3 Three Levels of "Activity"

There can be different levels at which we can describe an activity. For example, we can imagine a three-level description [Dietterich and Bui, 2006]. At the highest level is the kind of weak activity model that TaskTracer has: an activity is a collection of "resources" (web pages, folders, documents, email messages, email contacts). Typical activities are "Advising student YY", "Project ZZ", and so on. They are called *tasks* in TaskTracer. We discuss how to recognize such activities in Chapter 2 and 3.

The next level is a kind of workflow model. A *workflow* can be thought of as a "To-Do" item, consisting of several steps with some temporal constraints. A workflow usually involves an initiation (typically the receipt of an email message with some information such as a URL or an attached file and a request or command), a partially-ordered sequence of steps, and a termination. The "task" above usually consists of several workflows. We will describe some workflow examples and discuss how to recognize them in Chapter 4 .

The third level is the functional *operation*, such as retrieving a paper, saving a file in the "reading" folder, and printing a copy of a file, and so on. A workflow usually consists of a bunch of operations. In this thesis, we will concentrate on the higher levels of activity – "task" and "workflow".

An example of an activity hierarchy is depicted in Figure 1.3. The user has several tasks, such as *advising student Shen* and *project TaskTracer*. Each task also involves several workflow procedures. For example, the task *advising student Shen* involves the workflow *comments on thesis*, *scheduling meeting*, and so on. Each workflow procedure consists of several functional operations. For example, the workflow *comments on thesis* consists of the functional operation *retrieve thesis*, *write notes*, and so on.

Our research goal is to provide appropriate assistance to the user. Thus, we focus on recognizing the higher-level activities – tasks and workflows. As we will discuss later, recognizing such activities has the potential to greatly improve the efficiency of knowledge workers.

## 1.4 Thesis Outline

The dissertation is structured as follows. Chapter 2 gives some background about our research and presents our first version of the task recognition system, which is based on batch learning. Chapter 3 analyzes the shortcoming of the previous system and presents a novel online learning system to detect task switches. The new system has a more friendly user interface supporting negotiated interruption. Section 4 introduces the workflow recognition problem. We mine the workflow models from the user's TaskTracer event log and apply such mined models to workflow recognition. We conclude the thesis with a summary of contributions and a disscusion of the future work.

# Chapter 2 – Single Window Classifier and Its Extension for Task Recognition

The TaskTracer system seeks to help multi-tasking users manage the resources that they create and access while carrying out their work tasks. It does this by associating with each user-defined task the set of files, folders, email messages, contacts, and web pages that the user accesses when performing that task. The initial TaskTracer system relies on the user to notify the system each time the user changes tasks (See Figure 1.2). However, this is burdensome, and users often forget to tell TaskTracer what task they are working on.

This chapter introduces TaskPredictor [Shen *et al.*, 2006], a machine learning system that attempts to predict the user's current task. We first present the single window classifier, which 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. We then describe how to combine multiple single-window predictions (specifically, the class probability estimates) to obtain more reliable predictions [Shen *et al.*, 2007]. We study 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.

## 2.1 Overview

Today's desktop information workers continually switch 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, and so on). 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.

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 [Fawcett and Provost, 1999; Haigh and Yanco, 2002]. Some commercial applications have been developed [Horvitz *et al.*, 1998]. 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.

Horvitz [1999] proposes several critical factors for the effective integration of automated services into user interfaces, which include:

1. **Tailoring the service to match uncertainty, variation in goals**. A preference for "doing less" but doing it correctly under uncertainty can provide a valuable advance towards a solution and minimize the need for costly

undoing or backtracking.

2. **Maintaining working memory of recent interactions**. Systems should maintain a memory of recent interactions with users and provide mechanisms that allow users to make efficient and natural references to objects and services.

3. **Continuing to learn by observing**. Automated services should be endowed with the ability to continue to become better at working with users by continuing to learn about a user's goals and needs.

4. **Considering the status of a user's attention in the timing of services**. Intelligent assistants should employ models of the attention of users and consider the costs and benefits of deferring the action to a time when the action will be less distracting.

TaskPredictor was designed to address the above factors. We will discuss its detailed design in the rest of this thesis.

## 2.2 A Hybrid Learning System for Task Recognition

TaskPredictor predicts the current task based on properties of the window currently in focus. In order to "do less" but do it correctly, TaskPredictor only makes inferences based on "informative" observations and only issues a switch alert when its confidence is larger than a specified threshold. When TaskPredictor is confident about the switch prediction, it pops up a balloon to remind the user.

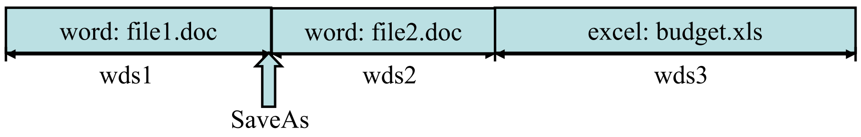


Figure 2.1: Examples of WDSs. Switching to another application or another document will create a new WDS. There are three WDSs in our example.

### 2.2.1 Abstracting Observations from the Event Stream

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. [2005].

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:

- **Event type** — such as window focus, file open, file save, web page navigation, and so on;
- **Listener ID** — the source of the EventMessage (MS Office, file system hook, Internet Explorer, Windows explorer, and so on);
- **Body** — the detailed information about the event e.g., pathname, window title, email address, Uniform Resource Locator (URL);
- **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. An example is shown in Figure 2.1.

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 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 $j$. If this word appears in the event, $x_j$ is 1, otherwise $x_j$ is 0.

### 2.2.2 Machine Learning Methods

It is important for an intelligent system to tailor its service to match its uncertainty. A preference for "doing less" but doing it correctly under uncertainty can provide a valuable advance towards a solution and minimize the need for costly undoing or backtracking. To achieve this goal, we employ the following three machine learning methods: (a) classification thresholds, (b) mutual information feature selection, and (c) a hybrid Naive Bayes/Support Vector Machine classifier.

#### 2.2.2.1 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 $\theta$. 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 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 $\theta$, and if $P(\mathbf{x}) > \theta$, it makes a prediction. Otherwise, it does nothing. In effect, TaskPredictor 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.

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, 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 should catch this promptly and make a correct prediction for at least one WDS in the episode.

#### 2.2.2.2 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 [Porter, 1980]. For example, this converts "tracing", "traced", and "tracer" to the root word "trace" [Porter, 1980].

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 [Yang and Pedersen, 1997]. 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 on an ordinary PC computer ($Pentium4$ CPU, 512MB).

#### 2.2.2.3 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 [Joachims, 2001]. To apply SVMs in our multi-class 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. [2004] 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 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 $P_{NB}(\mathbf{x})$. We compare $P_{NB}(\mathbf{x})$ to $\theta$ and make a prediction using the learned SVM if $P_{NB}(\mathbf{x}) > \theta$. 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).

### 2.2.3 Experimental Results

We deployed TaskTracer on Windows machines in our research group. To evaluate TaskPredictor, we required a dump of the TaskTracer database, which raises many privacy concerns. As a result, we only obtained data from 2 professors. We refer to them as FA and FB (the faculty members). Subject FB manually reviewed his data and cleaned it so that it contained more accurate task labels. The collected data is summarized in Table 2.1.

Table 2.1: Datasets for Evaluating TaskPredictor (number of words is computed after stoplist and stemming)

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

#### 2.2.3.1 Evaluation Methodology

For TaskPredictor, 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.

#### 2.2.3.2 Effect of Feature Selection

We first assess the value of performing feature selection. Figures 2.2 and 2.3 show the results of feature selection for FA and FB on TaskPredictor. 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 $\theta$ to maximize the precision, subject to the constraint that $\theta$ should be large enough that the learning algorithm makes at least 100 correct predictions.

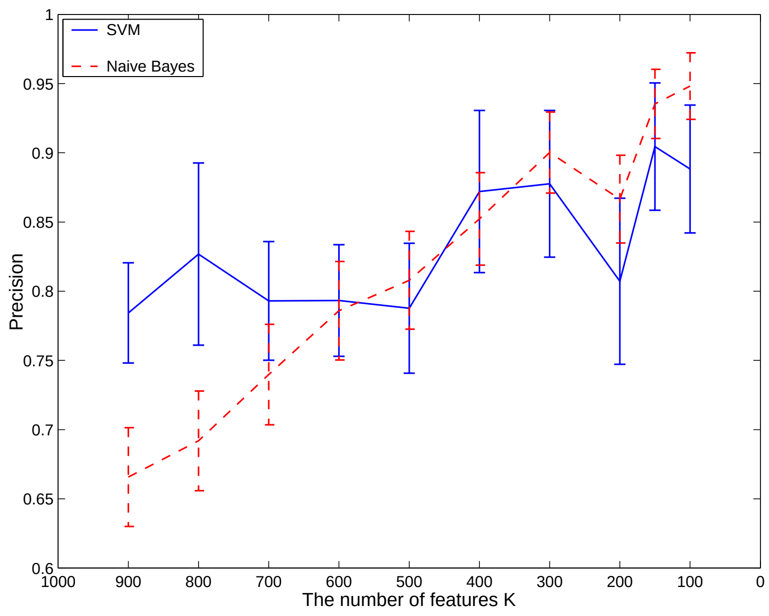


Figure 2.2: Precision of TaskPredictor for FA as a function of the number of selected features $K$. Error bars denote 95% confidence intervals.

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.

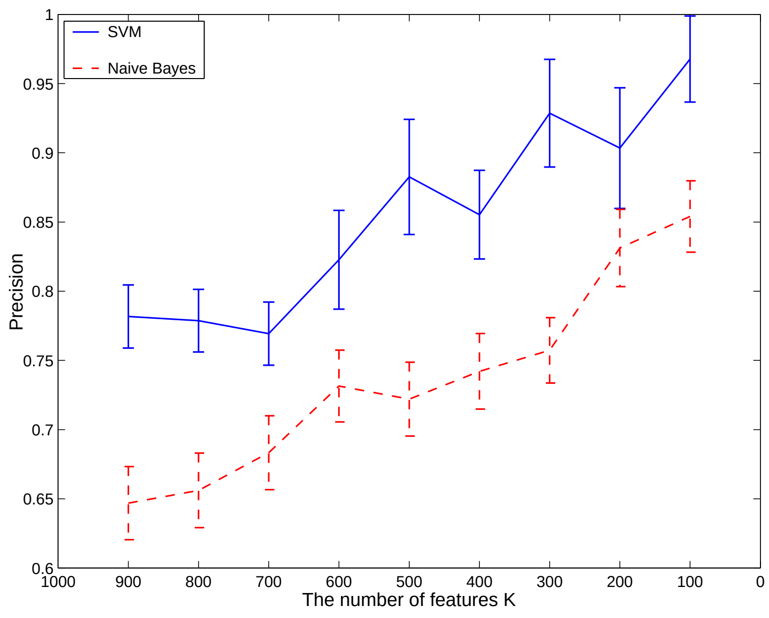


Figure 2.3: Precision of TaskPredictor for FB as a function of the number of selected features $K$. Error bars denote 95% confidence intervals.

#### 2.2.3.3 Effect of the classification threshold

Now we analyze the effect of the classification threshold. Figures 2.4 and 2.5 show the precision of TaskPredictor 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 $\theta$ is implicit in these plots. Low values of $\theta$ correspond to high coverage, and high values of $\theta$ give low coverage. We see that as we increase $\theta$, the precision increases very well. TaskPredictor 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.

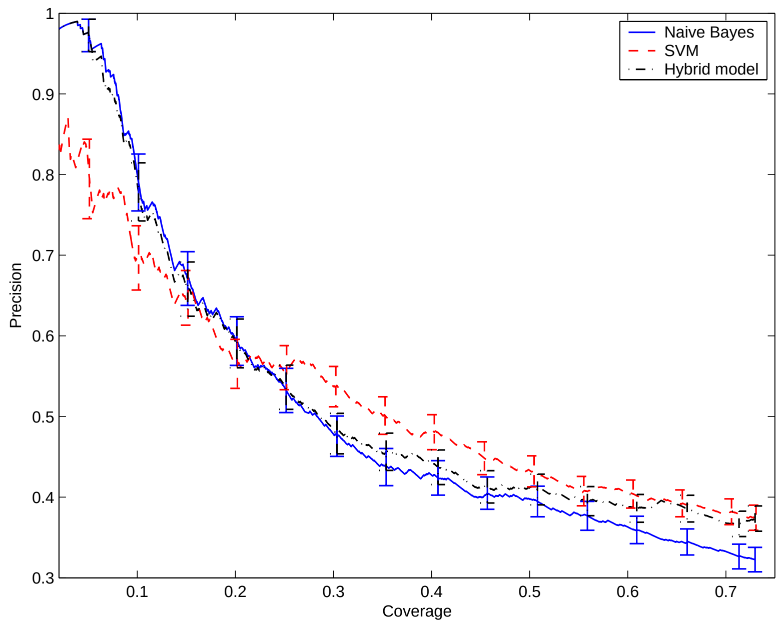


Figure 2.4: Precision of TaskPredictor as a function of the coverage for FA, created by varying $\theta$.

### 2.2.3.4 Effect of the Hybrid Method

The preceding figures also show the effect of the Hybrid method. For FA (Figure 2.4), 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 2.5), 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.

These results show that the hybrid method gives performance equal to or better than the best single method (Naive Bayes or SVMs). This supports our hypothesis that using a generative model to decide when to make predictions works better

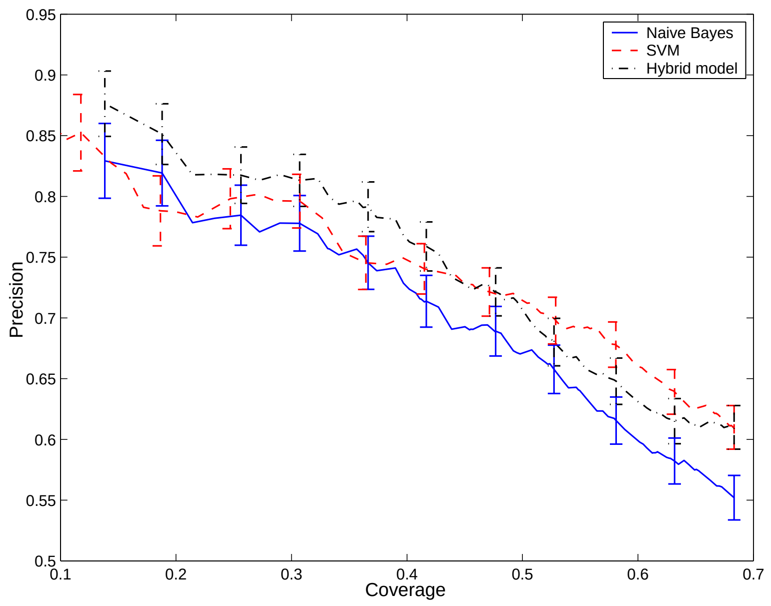


Figure 2.5: Precision of TaskPredictor as a function of the coverage for FB, created by varying $\theta$.

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.

#### 2.2.3.5 Online Performance

It is interesting to plot the cumulative online performance of the learning algorithms over the evaluation period. Figure 2.6 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 $\theta$. 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

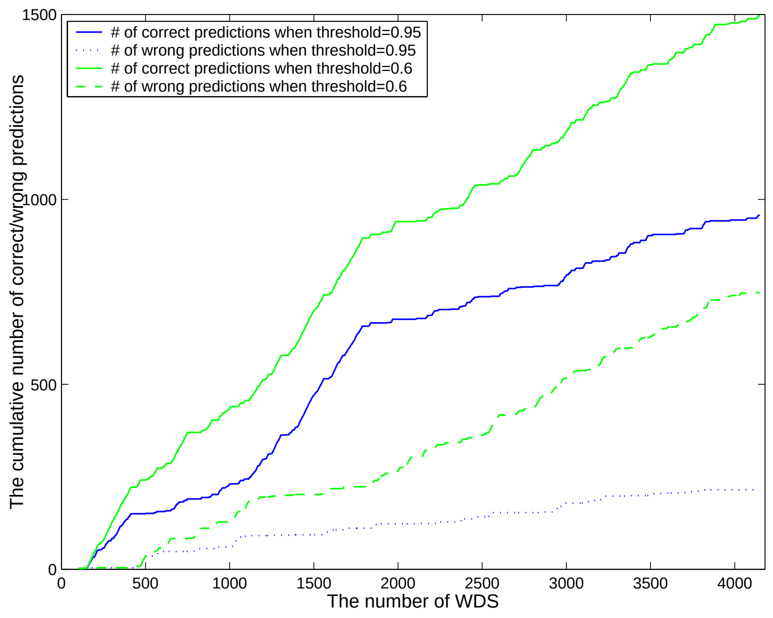


Figure 2.6: Cumulative correct and incorrect predictions for FB as a function of the number of WDSs processed (the hybrid model)

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 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.

### 2.2.4 Conclusions

This section has presented a learning system for predicting the current task of the user. The TaskPredictor system predicts the user's current TaskTracer task based on the title and document pathname (or URL) of the window currently in focus.

We demonstrated that three machine learning techniques gave improved performance with the system: 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 a TaskPredictor that is sufficiently accurate to be useful.

## 2.3 Combining Predictions from Multiple Observations

The TaskTracer system provides a convenient, low-cost way for knowledge workers 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. In this section, 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 section 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.

### 2.3.1 Introduction

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.

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, and so on without remembering to first inform TaskTracer. In such cases, we would like TaskTracer to automatically detect the task 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 thesis, 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 section 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 section is structured as follows. First, we briefly describe the Single Window Classifier. 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 section. Third, we show experimental results on both synthetic and real user data. We conclude the section with a review of related work and a discussion of future work.

### 2.3.2 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, we create multiple training instances for each WDS. For a WDS that lasts $t$ seconds, we generate $n = \lceil \frac{5}{1+5\exp(-0.1t)} \rceil$ 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 .

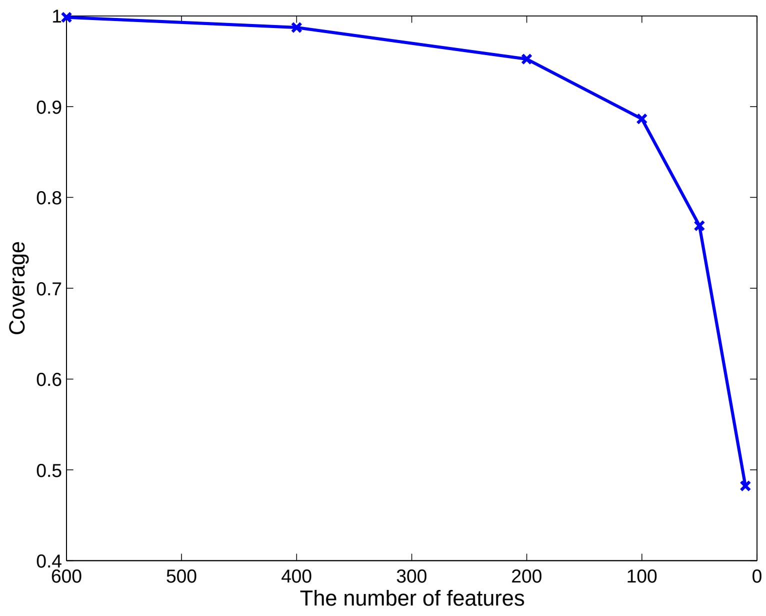


Figure 2.7: Impact of feature selection. Fraction of one-minute intervals that have informative observations as a function of the number of selected features.

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 2.7 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.

### 2.3.3 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 $\ell$ observations $\{\mathbf{x}_{t+1-\ell}, \ldots, \mathbf{x}_t\}$. The larger value of $\ell$ 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.

#### 2.3.3.1 A Framework to Detect Task Switches

The above algorithm 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 $\ell$. Once $\mathbf{X}$ contains $\ell$ elements, each new element added

**Algorithm 2.1** Generic framework for detecting task switches.

```
Require: $\ell$ – we will store $\ell$ observations in queue $\mathbf{X}$
Require: $d$ – efficiency delay; skip switch computation if WDS has lasted longer than $d$ seconds
Require: $r$ – recovery time; keep silent for the next $r$ informative observations after a switch alarm
 1: NoEventTime $\Leftarrow 0$
 2: SilentNumber $\Leftarrow \infty$
 3: repeat {every $f$ seconds or if an event happens}
 4:   if no event happened then
 5:     NoEventTime $\Leftarrow$ NoEventTime $+f$
 6:   else
 7:     NoEventTime $\Leftarrow 0$
 8:   end if
      //skip inference if nothing has happened for $d$ seconds
 9:   if NoEventTime $> d$ then
10:     continue
11:   end if
      //update the observation queue $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$
12:   $\mathbf{x}_{now} \Leftarrow$ GetObservation()
13:   if $\mathbf{x}_{now}$ is not null then
14:     UpdateQueue($\mathbf{X}, \mathbf{x}_{now}, \ell$)
15:     SilentNumber $\Leftarrow$ SilentNumber $+1$
16:   else
17:     continue
18:   end if
19:   if SilentNumber $\leq r$ then
20:     continue
21:   end if
22:   ($\Lambda$, PredictedTask) $\Leftarrow$ SwitchMetric ($\mathbf{X}$)
23:   if $\Lambda >$ threshold then
24:     SilentNumber $\Leftarrow 0$
25:     if PopUpAlert(PredictedTask)==OK then
26:       SetCurrentTask(PredictedTask)
27:     end if
28:   end if
29: until the system is shutdown
```

to **X** causes the oldest element to be removed. The function SwitchMetric computes the switch metric value $\Lambda$ from **X** and also returns the predicted new task as discussed below. If $\Lambda$ 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 the cost of computing the switch metric. We do not want **X** to be full of identical observations from the same long-duration WDS. Hence, there is no need to update **X** either.

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

#### 2.3.3.2 Metric Functions

We now describe three methods for computing metric functions based on the FIFO queue of informative observations **X**. First we describe the baseline method, which is similar to what has been deployed in TaskPredictor.

**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

$$\Lambda = I[y_a \neq y_b] \cdot \left\{ \sum_{i=l}^{\ell/2} P(y_a|\mathbf{x}_i) + \sum_{i=\ell/2+1}^{\ell} P(y_b|\mathbf{x}_i) \right\}, \tag{2.1}$$

where $I[p] = 1$ if $p$ is true and $-1$ otherwise. Throughout this section, 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 $\Theta_i$ be the posterior probability distribution over the tasks given observation $\mathbf{x}_i$, and $\mu_a$, $\mu_b$ 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 $\Lambda$ as the Gaussian likelihood ratio score

$$\Lambda = (\overline{\Theta}_a - \overline{\Theta}_b)^T \left(\frac{2}{\ell}\Sigma_a + \frac{2}{\ell}\Sigma_b\right)^{-1} (\overline{\Theta}_a - \overline{\Theta}_b), \tag{2.2}$$

where $\overline{\Theta}_a$ and $\overline{\Theta}_b$ are the sample means and $\Sigma_a$ and $\Sigma_b$ 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 $\Lambda$. If we assume $\forall i \neq j$, $y_i$ and $y_j$ are independent, and treat $\Sigma_a$ and $\Sigma_b$ 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 $\ell$ should be quite large (e.g., $> 30$). However, very large $\ell$ 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

$$C(\mathbf{Y}|\mathbf{X}) = \sum_{1 \leq i \leq \ell} C_a(y_i|\mathbf{x}_i) + \sum_{1 < i \leq \ell} C_t(\mathbf{x}_i|\mathbf{x}_{i-1}), \tag{2.3}$$

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:

$$\alpha = \min_a C(y_1 = y_2 = \ldots = y_\ell = a|\mathbf{X}), \tag{2.4}$$

$$\beta = \min_{i, a \neq b} C(y_1 = \ldots = y_i = a, y_{i+1} = \ldots = y_\ell = b|\mathbf{X}). \tag{2.5}$$

We can apply the Viterbi algorithm to find these minimal costs. Given $(\mathbf{x}_1, \ldots, \mathbf{x}_i)$, let $\alpha_{ij}$ be the minimal cost if we predict the task at time $i$ as $y_j$ and predict no switch until time $i$. Let $\beta_{ij}$ 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 $\alpha_{ij}$ 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 $\beta_{ij}$. Thus, $\forall 1 < i \leq \ell$, we can recursively calculate the cost for each $y_j$ at time $i$ as

$$\alpha_{ij} = \alpha_{(i-1)j} + C_a(y_j|\mathbf{x}_i), \tag{2.6}$$

$$\beta_{ij} = \min\{\beta_{(i-1)j}, \min_{k \neq j} \alpha_{(i-1)k} + c\} + C_a(y_j|\mathbf{x}_i). \tag{2.7}$$

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:

$$\Lambda = \alpha - \beta. \tag{2.8}$$

This naive Viterbi approach 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.

Since we assume a constant transition cost, we can adopt the forward-backward algorithm [Felzenszwalb *et al.*, 2004] to reduce the complexity to $O(\ell \cdot |y|)$.

Without the constraint on the number of transitions, $c_t(y_j)$, the minimal cost of labeling the observation $x$ as $y_j$ at time $t$ given the observations seen so far can be computed as

$$c_t(y_j) = c(y_j|x) + \min_i(c_{t-1}(y_i) + c(y_j|y_i)). \tag{2.9}$$

The main computation lies in $D(j) = \min_i(c_{t-1}(y_i) + c(y_j|y_i))$, which will be $O(n^2)$ if we use Viterbi to compute $D(j)$ for each $j$. With the $L1$ norm distance cost for $c(y_j|y_i)$, the method introduced in [Felzenszwalb *et al.*, 2004] can reduce the complexity of $D(j)$ to $O(n)$. In that case, the cost function looks like a "$\vee$", and the cost keeps linearly increasing. In our case, we assume a constant transition cost $c(y_t|y_{t-1}) = c \cdot \delta[y_t \neq y_{t-1}]$, where $c$ is a real and $\delta[p] = 1$ if $p$ is true and 0 otherwise. The cost function looks like a "$^-\vee^-$", and the cost will be the same after we move away from the center. See an example in Figure 2.8.

We can modify the approach with the $L1$ norm cost function to fit our requirement. To apply the forward-backward algorithm [Felzenszwalb *et al.*, 2004], we need to do a minimum filtering: we track $\beta$, which is the smallest cost assuming a transition seen so far. Let $C_j$ be the minimal cost found so far for task $y_j$. First, we make a forward pass through all tasks. For each task $y_j$, we compute $\beta = \min(\beta, c_{t-1}(y_{j-1}) + c)$, that is, the minimal cost seen so far either remains the same, or, takes the previous minimal cost of the previous task plus a transition cost (which is the flat plane of "$^-\vee^-$"). Then $C_j = \min(\beta, c_{t-1}(y_j))$, that is, it is either the minimal cost assuming a transition or the cost without a transition.

In the backward process, we gather the influence from the right to the left and

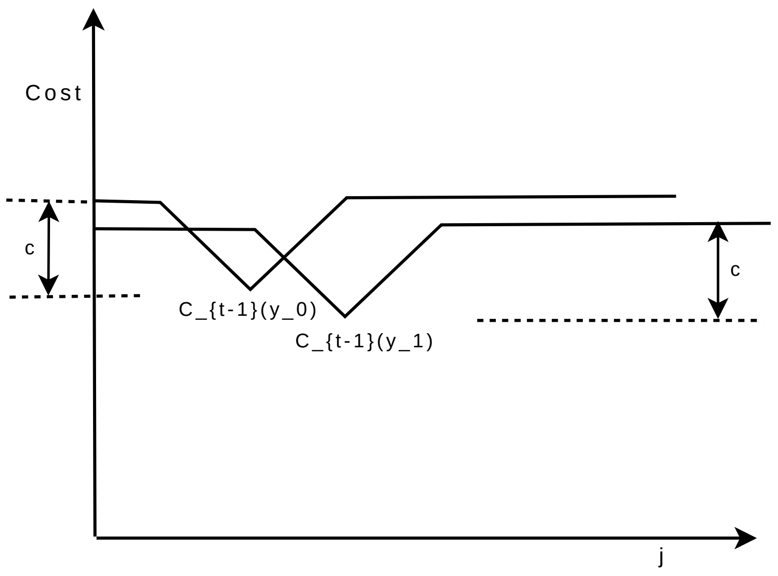


Figure 2.8: An example of our cost function.

update the values computed in the forward process. This time, for each task $y_j$ we compute $\beta = \min(\beta, c_{t-1}(y_{j+1}) + c)$. We let $C_j = \min(\beta, C_j)$. Then $D(j) = C_j$. The forward and backward passes take $O(n)$, thus we get an $O(Tn)$ algorithm for computing the minimum cost label sequence.

A small example is presented in Table 2.2 and Table 2.3.

| | | | | | |
|---|---|---|---|---|---|
| $c_{t-1}(y_j)$ | 3 | 1 | 4 | 2 | 1 |
| $c_{t-1}(y_{j-1}) + c$ | $\infty$ | 4 | 2 | 5 | 3 |
| $\beta$ | $\infty$ | 4 | 2 | 2 | 2 |
| $C_j$ | 3 | 1 | 2 | 2 | 1 |

Table 2.2: Forward pass (from the left to the right), assuming the transition cost $c = 1$.

The result $(2, 1, 2, 2, 1)$ is exactly what we will get if directly computing $\min_i(c_{t-1}(y_i) + c(y_j|y_i))$ for each activity $y_i$.

| | | | | | |
|---|---|---|---|---|---|
| $c_{t-1}(y_j)$ | 3 | 1 | 4 | 2 | 1 |
| $c_{t-1}(y_{j+1}) + c$ | 2 | 5 | 3 | 2 | $\infty$ |
| old $C_j$ | 3 | 1 | 2 | 2 | 1 |
| $\beta$ | 2 | 2 | 2 | 2 | $\infty$ |
| $C_j$ | 2 | 1 | 2 | 2 | 1 |

Table 2.3: Backward pass (from the right to the left), assuming the transition cost $c = 1$.

### 2.3.4 Experimental Results

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

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.

Real user data was obtained by deploying TaskTracer on Windows machines in our research group and collecting data from four users, whom we refer to as FA, FB, FC, and SA. The collected data is summarized in Table 2.4. 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.

Table 2.4: 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 |

**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 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

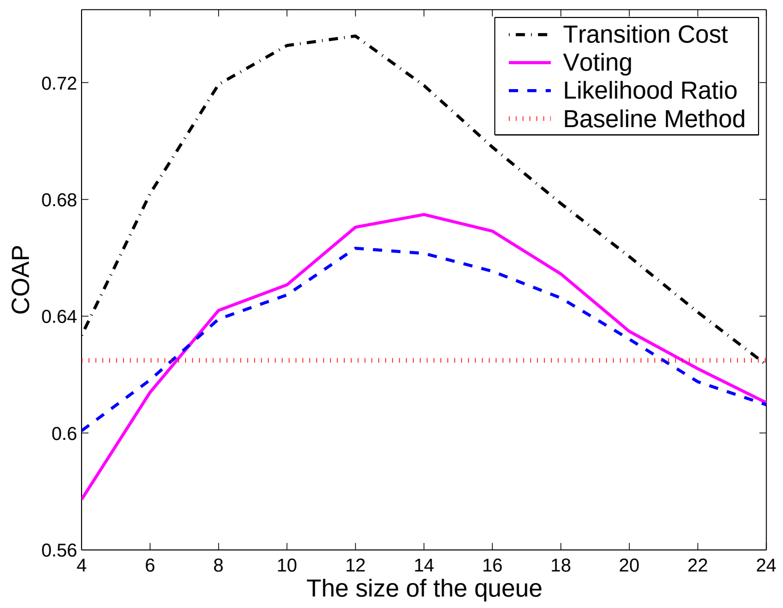


Figure 2.9: The average COAP values of the synthetic data as a function of $\ell$, the length of the observation queue $\mathbf{X}$

using A+B and apply the learned SWC and threshold value to evaluate on C. The results on C are plotted in Figure 2.9 and 2.10.

From the plots, $\ell$ (the size of informative observation queue) is crucial to the accuracy of methods. When $\ell$ 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 $\ell$ is large. The maximal COAPs are reached when $\ell$ 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

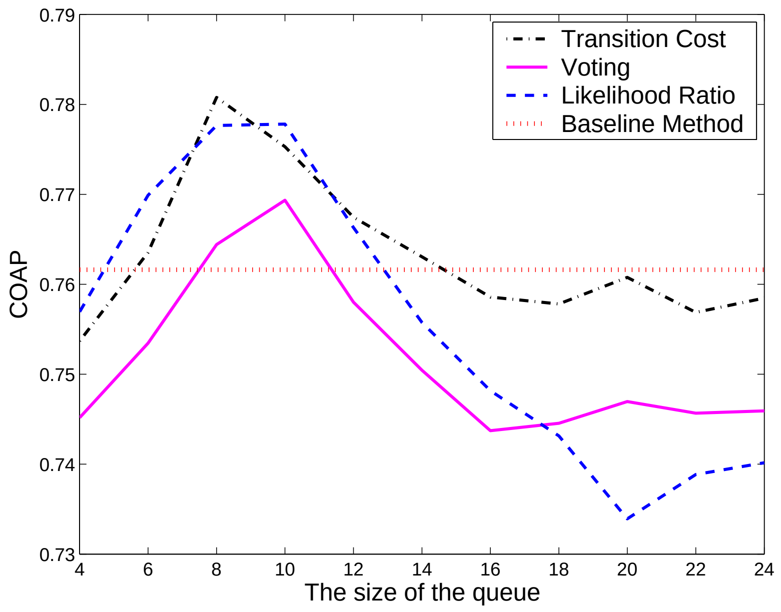


Figure 2.10: The average COAP values of the real user data as a function of $\ell$

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

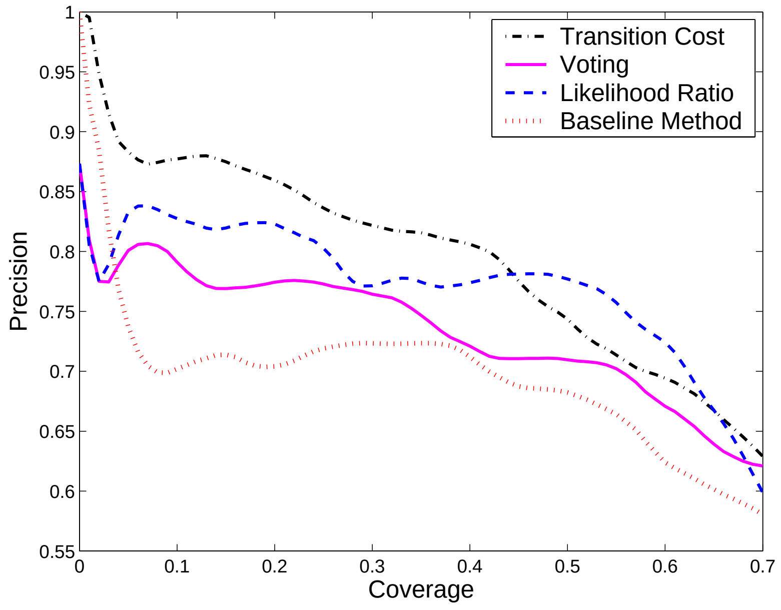


Figure 2.11: The average precision values as a function of the coverage for real users given 10 observations, created by varying the prediction threshold.

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 assigned $d_0$ 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 2.11.

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.

### 2.3.5 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 the 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 tasks 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.

### 2.3.6 Conclusions

In this section, 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.

# Chapter 3 – Efficient Online Learning for Task Recognition

The TaskTracer system allows knowledge workers to define a set of tasks that characterize their desktop work. It then associates with each user-defined task the set of resources that the user accesses when performing that task. In order to correctly associate resources with tasks and provide useful task-related services to the user, the system needs to know the current task of the user at all times. It is often convenient for the user to explicitly declare which task he/she is working on. But frequently the user forgets to do this. TaskTracer applies machine learning methods to detect undeclared task switches and predict the correct task of the user. This chapter presents *TaskPredictor2* [Shen *et al.*, 2009b], a complete redesign of the task predictor in TaskTracer and its notification user interface. TaskPredictor2 applies a novel online learning algorithm that is able to incorporate a richer set of features than our previous predictors. We prove an error bound for the algorithm and present experimental results that show improved accuracy and a 180-fold speedup on real user data. The user interface supports negotiated interruption and makes it easy for the user to correct both the predicted time of the task switch and the predicted task.

## 3.1 Overview

The TaskTracer system [Dragunov *et al.*, 2005; Shen *et al.*, 2006] is an intelligent activity management system [Kaptelinin, 2003; Quan and Karger, 2003; Bellotti *et al.*, 2003; Geyer *et al.*, 2006] that helps knowledge workers manage their work. TaskTracer then configures the desktop in several ways to support the user. In order to automatically associate resources with tasks, TaskTracer needs to know the current task of the user. TaskTracer provides a taskbar component, the Task Selector, that makes it easy for the user to explicitly declare the current task. However, when the user forgets to do this, resources become associated with incorrect tasks. This will cause errors in the resources displayed in Task Explorer, incorrect folder predictions, and errors in time reporting.

To address this problem, we have previously developed two machine learning methods for detecting task switches and predicting the current task of the user, which are described in Chapter 2. The first of these was the "single window classifier" that based its predictions only on the file path (or URL) of the current resource and the title of the active window. An advantage of this was that the predictions could be computed immediately and rapidly each time the user switched focus from one window to another. The disadvantage was that with such impoverished features, the accuracy of the predictions was not very high.

The second learning method was based on analyzing the *sequence* of recently-visited resources. It first applied the single-window classifier to each of these resource visits. Then it detected task switches by applying a Viterbi algorithm with a fixed switch cost. This led to somewhat improved accuracy but had the side effect of introducing a potentially significant delay between the time of the switch and the time the switch was detected. This led in turn to major usability problems. The UI that we developed for notifying the user consisted of a pop-up window. By the time the switch was detected, the user was typically deeply engaged in the new task, so that the pop-up window created an expensive interruption. Furthermore, the pop-up window UI only allowed the user to accept or correct the prediction. However, if the true switch occurred at time $t$ and the prediction was accepted at time $t'$, then all resources visited between $t$ and $t'$ would be incorrectly associated with the old task. There was no way—short of going to Task Explorer and dragging the incorrectly-associated resources to the right task—to fix these errors.

An additional drawback of both of these learning methods is that they employed a batch SVM algorithm [Joachims, 2001; Chang and Lin, 2001]. This must store all of the training examples and reprocess all of them when retraining is required. SVM training time is roughly quadratic in the number of examples, so the longer TaskTracer is used, the slower the training becomes. Furthermore, task prediction is a multi-class prediction problem over a potentially large number of classes. For example, our busiest user worked on 299 different tasks over a four-month period. To perform multi-class learning with SVMs, we employed the one-versus-rest approach. If there are $K$ classes, then this requires learning $K$ classifiers. If there are $N$ examples per class, then the total running time is roughly $O(N^2K^3)$. This is not practical in an interactive, desktop setting.

The goal of the redesign of the TaskPredictor (and of this chapter) was to address these four problems:

- Accuracy: How can we improve the accuracy of the task switch predictor?
- Memory/CPU Cost: Can we switch to a more efficient learning algorithm that requires constant memory and CPU time to train and make predictions?
- Prediction Delay/Interruption Cost: Can we develop a user interface that is able to manage the delay between the time a switch occurs and the time it is predicted? How can this UI minimize interruption costs?
- Retroactive Association Changes: When the user changes the time of an task switch, how can that change be reflected in the associations between resources and tasks?

This chapter presents *TaskPredictor2*, a new switch detection system that fixes all of these problems.

## 3.2 System Design

TaskPredictor2 captures a set of richer contextual information and provides a better user experience. It consists of the following components (see Figure 3.1):

- The **Association Engine** manages the associations between resources and tasks. Each time the user visits a resource, an event is triggered. In response, the Association Engine starts a 20-second timer. If the resource is still in focus after 20 seconds and if it has not been previously associated with the current declared task, then it is automatically associated with the current declared task. In addition to recording the association in a data base table, the Association Engine also places an entry in an Association History database table that keeps track of the time the association was established, the id of the resource, and the id of the task. When the user accepts or corrects a task switch, the UI sends an "AssociationHistoryRevisions" event specifying the time interval that is affected by the change and the id of the task that now should be associated with this interval. The Association Engine then revises the associations for this time interval.

- The **State Estimator** monitors desktop events. Once each minute, it computes an information vector that summarizes the state of the desktop and sends this to the Switch Detector.

- The **Switch Detector** analyzes the information vectors provided by the State Estimator and converts them into a collection of feature vectors as described below. These are processed by the learned classifier to predict whether there has been an undeclared task switch and (if so) the time of the switch, the id of the new task, and the prediction confidence. This is passed to the Notification Controller and the UI.

- The **Notification Controller** chooses when and how to notify the user with the goal of minimizing both the interruption cost and association errors. When the system becomes confident that the user forgot to declare a task switch, the user might be busy with something. Notifying the user right then

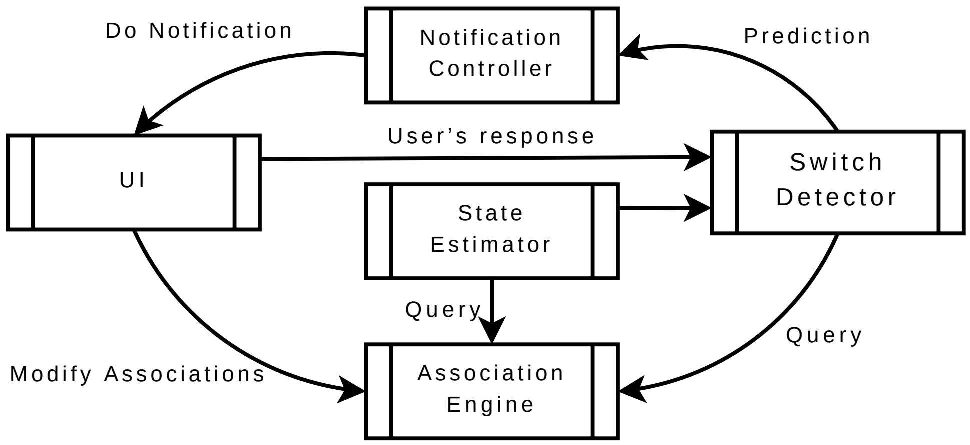
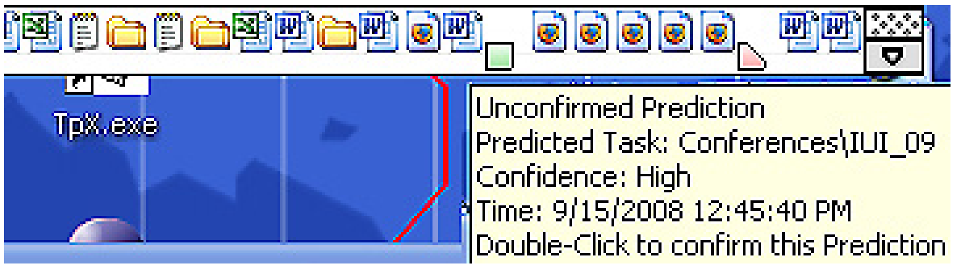


Figure 3.1: The system architecture of the task recognition system. Arrows indicate information flows.

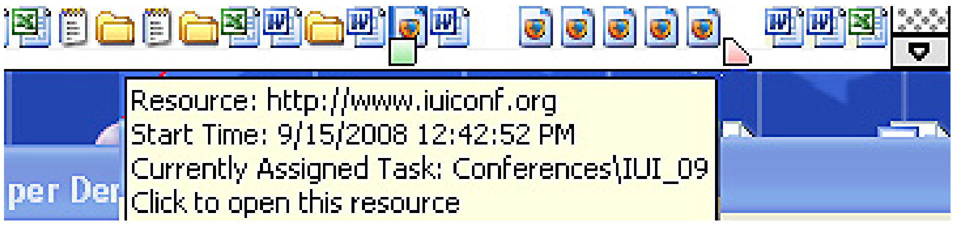
would have a high interruption cost.

- The **UI** presents the switch notification to the user and provides the user a variety of ways to respond to (and correct) the predicted switch. It provides feedback to the Switch Detector so that it can retrain its predictors.

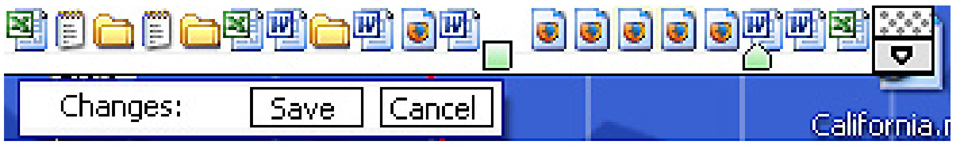
The current version of the Notification Controller employs a simple heuristic to determine when to issue the switch alert: If the system is confident that a task switch has occurred and the user is at an operation boundary and likely has low interruption cost (e.g., opening a file dialog box or switching focus to another window), then an alert should be given. Following the design guideline suggested by Gluck et al. [2007], the alert takes three forms depending on the confidence of the predictor: (a) displaying a red switch icon (see Figure 3.2), (b) shaking the timeline UI for one second, or (c) opening a pop-up window. The rationale is that highly confident predictions are more likely to be correct, so ignoring them is more likely to lead to association errors.


Unconfirmed Prediction
Predicted Task: Conferences\IUI_09
Confidence: High
Time: 9/15/2008 12:45:40 PM
Double-Click to confirm this Prediction

(a) Notification Controller issues a switch alert (the red triangle) + tool tip.


Resource: http://www.iuiconf.org
Start Time: 9/15/2008 12:42:52 PM
Currently Assigned Task: Conferences\IUI_09
Click to open this resource

(b) Hovering the mouse over a resource icon gives more information.



(c) If a switch is changed, UI will ask for confirmation (the Save/Cancel box).

Figure 3.2: Interactions between TaskPredictor2 and the user.

Figure 3.2 shows an example of the interactions between UI and the user. The UI consists of two rows of icons. The upper row displays one icon for each time that a resource was visited by the user (in the order that these visits occurred). Multiple visits result in multiple icons—however, very brief visits are ignored. Information about each resource (path name/URL, associated task) is shown as a tool tip if the user hovers the mouse over the resource icon (see Figure 3.2b). The user can also double-click on an icon to cause that resource to be opened and brought into focus.

The lower row displays one icon for each task switch. The shape and color of the icon indicates the type of switch. User-declared switches are displayed as green boxes. Unconfirmed switch predictions are displayed as red triangles. Confirmed switch predictions are displayed as a green "house". A tool tip shows the task name, the type of switch, and the time of the switch. The user can drag and drop a switch icon to change the switch time. However, we do not permit the drag operation to go past another existing switch time.

In the examples shown in Figure 3.2, the user first worked on the task *IUI-09*. Then he explicitly switched to *Reading News* and browsed some webpages. The green box indicates this switch. After reading the news, he resumed working on task *IUI-09*. Unfortunately he forgot to explicitly indicate that switch. TaskPredictor2 detected this undeclared switch and decided that it was the right time to notify the user. So it drew a red triangle to indicate this switch and shook the UI to attract the user's attention. The user then hovered the mouse over the triangle to see the predicted task (Figure 3.2a). To confirm that the prediction is correct, the user can double-click the switch icon. The icon then turns into a "house" (see Figure 3.2c). If the prediction is wrong (i.e., there was no switch at all), the user can right click and select "Remove" from a pop-up menu. If the predictor has correctly identified a task switch but predicted the wrong task, the user can right click and choose "Change the Task". This brings up a Task Chooser UI. If the switch is predicted at the wrong time point, the user can drag the switch icon to the correct point in time. As the icon moves, the UI provides tool tips to show the full name of the resource, the time it was accessed, and the task to which it is currently assigned so that the user can find the right switch point. In all cases, the UI requests confirmation ("Save") or allows the user to cancel out of the changes ("Cancel"). If the user ignores a notification, TaskPredictor2 does nothing. In most cases, if the user ignores a notification long enough, it will disappear because the user will continue to accumulate time on an open resource, and this resource will become associated with the current declared task. This will, in turn, cause TaskPredictor2 to stop predicting that a task switch has occurred, because it now has evidence that the open resource is associated with the declared task.

As in our earlier method, there can be a substantial delay between the time the user switches tasks and the time TaskPredictor2 detects this. A key design goal of this UI was to support this by providing asynchronous (negotiated) interruption and retroactive correction of incorrect associations. After the user confirms a change, the information is sent to the Association Engine to update the resource-task associations.

## 3.3 Efficient Learning Algorithms for Task Recognition

To address the accuracy and CPU cost of our previous predictors, we adopted an efficient online learning algorithm and provided it with a richer set of features. Online learning algorithms have been shown to be efficient in many large-scale problems. The algorithm is based on the Perceptron family of algorithms [Rosenblatt, 1988], and, particularly, on the PA algorithm [Crammer *et al.*, 2006].

### 3.3.1 Feature Design and Scoring Function

The State Estimator monitors various desktop events (Open, Close, Save, SaveAs, Change Window Focus, and so on). Every $s$ seconds (default $s$=60) or when the user declares a task switch, it computes an information vector $\mathbf{X}_t$ describing the time interval $t$ since the last information vector was computed. This information vector is then mapped into feature vectors by two functions: $\mathbf{F}_T : (\mathbf{X}_t, y_j) \rightarrow R^k$ and $\mathbf{F}_S : (\mathbf{X}_t) \rightarrow R^m$. The first function $\mathbf{F}_T$ computes *task-specific* features for a specified task $y_j$; the second function $\mathbf{F}_S$ computes *switch-specific features*. The task-specific features include

- Strength of association of the active resource with task $y_j$: if the user has explicitly declared that the active resource belongs to $y_j$ (e.g., by drag-and-drop in TaskExplorer), the current task is likely to be $y_j$. If the active resource was implicitly associated with $y_j$ for some duration (which happens when $y_j$ is the declared task and then the resource is visited), this is a weaker indication that the current task is $y_j$.

- Percentage of open resources associated with task $y_j$: if most open resources are associated with $y_j$, it is likely that $y_j$ is the current task.

- Importance of window title word $x$ to task $y_j$. Given the bag of words $\Omega$, we compute a variant of TF-IDF [Joachims, 2001] for each word $x$ and task $y_j$:

$\text{TF}(x, \Omega) \cdot \log \frac{|\overline{S}|}{\text{DF}(x,\overline{S})}$. Here, $\overline{S}$ is the set of all feature vectors not labeled as $y_j$, $\text{TF}(x, \Omega)$ is the number of times $x$ appears in $\Omega$ and $\text{DF}(x, \overline{S})$ is the number of feature vectors containing $x$ that are not labeled $y_j$.

These task-specific features are intended to predict whether $y_j$ is the current task. The switch-specific features predict the likelihood of a switch. They include

- Number of resources closed in the last $s$ seconds: if the user is switching tasks, many open resources will often be closed.
- Percentage of open resources that have been accessed in the last $s$ seconds: if the user is still actively accessing open resources, it is unlikely there is a task switch.
- The time since the user's last explicit task switch: immediately after an explicit switch, it is unlikely the user will switch again. But as time passes, the likelihood of an undeclared switch increases.

To detect a task switch, we adopt a sliding window approach: at time $t$, we use two information vectors ($\mathbf{X}_{t-1}$ and $\mathbf{X}_t$) to score every pair of tasks for time intervals $t-1$ and $t$. Given a task pair $\langle y_{t-1}, y_t \rangle$, the scoring function $g$ is defined as

$$\begin{aligned} g(\langle y_{t-1}, y_t \rangle) &= \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_{t-1}, y_{t-1}) + \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_t, y_t) \\ &+ \phi(y_{t-1} \neq y_t) \left( \Lambda_2 \cdot \mathbf{F}_S(\mathbf{X}_{t-1}) + \Lambda_3 \cdot \mathbf{F}_S(\mathbf{X}_t) \right), \end{aligned}$$

where $\Lambda = \langle \Lambda_1, \Lambda_2, \Lambda_3 \rangle \in \mathbb{R}^n$ is a set of weights to be learned by the system, $\phi(p) = -1$ if $p$ is true and 0 otherwise, and the dot $(\cdot)$ means inner product. The first two terms of $g$ measure the likelihood that $y_{t-1}$ and $y_t$ are the tasks at time $t-1$ and $t$ (respectively). The third term measures the likelihood that there is no task switch from time $t-1$ to $t$. Thus, the third component of $g$ serves as a "switch penalty" when $y_{t-1} \neq y_t$.

We search for the $\langle \hat{y}_1, \hat{y}_2 \rangle$ that maximizes the score function $g$. If $\hat{y}_2$ is different from the current declared task and the score is larger than a specified threshold, then a switch prediction and the score are sent to the Notification Controller. At first glance, this search over all pairs of tasks would appear to require time quadratic in the number of tasks. However, the following algorithm computes the best score in linear time:

$$
\begin{aligned}
y^*_{t-1} &:= \arg\max_y \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_{t-1}, y) \\
A(y^*_{t-1}) &:= \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_{t-1}, y^*_{t-1}) \\
y^*_t &= \arg\max_y \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_t, y) \\
A(y^*_t) &= \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_t, y^*_t) \\
S &= \Lambda_2 \cdot \mathbf{F}_S(X_{t-1}) + \Lambda_3 \cdot \mathbf{F}_S(\mathbf{X}_t) \\
y^* &= \arg\max_y \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_{t-1}, y) + \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_t, y) \\
AA(y^*) &= \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_{t-1}, y^*) + \Lambda_1 \cdot \mathbf{F}_T(\mathbf{X}_t, y^*)
\end{aligned}
$$

Each pair of lines can be computed in time linear in the number of tasks. We assume $y^*_{t-1} \neq y^*_t$. To compute the best score $g(\langle \hat{y}_1, \hat{y}_2 \rangle)$, we compare two cases: $g(y^*, y^*) = AA(y^*)$ is the best score for the case where there is no change in the task from $t-1$ to $t$, and $g(y^*_{t-1}, y^*_t) = A(y^*_{t-1}) + A(y^*_t) + S$ if there is a switch. When $y^*_{t-1} = y^*_t$, we can compute the best score for the "no switch" case by tracking the top two scored tasks at time $t-1$ and $t$.

## 3.3.2 Regularized Passive-Aggressive Algorithm

In TaskPredictor2 we decided to adopt an error-driven online approach [Rosenblatt, 1988] to train $\Lambda$. Online learning algorithms only need to process the most recent observation to update the classifier. They have very limited requirements for CPU time and memory. Their efficiency has made them popular for many large-scale learning problems. An advantage of perceptron-style algorithms is that they are "conservative"—that is, they only retrain when an error is made. This further reduces the CPU cost of training, and it may also contribute to prediction accuracy by avoiding overfitting on data points that are already correctly classified.

We further chose to adopt a modification of the *Passive-Aggressive* (*PA*) algorithm [Crammer *et al.*, 2006]. The basic idea of this algorithm is that when an error is committed and feedback is received, the algorithm updates the weights as little as possible while ensuring that the error would not be committed again. More specifically, it chooses a step size that is exactly large enough to ensure that the classifier will correctly classify the erroneous case with a margin of 1. This is highly desirable in an intelligent user interface, because it avoids the problem where the user provides feedback but the algorithm only takes a small step in the right direction, so the user must repeatedly provide feedback until enough steps have been taken to fix the error. This behavior has been reported to be extremely annoying.

The standard Passive-Aggressive algorithm works as follows: Let the real tasks be $y_1$ at time $t-1$ and $y_2$ at time $t$, and $\langle \hat{y}_1, \hat{y}_2 \rangle$ be the highest scoring incorrect task pair. When the system makes an error, it would update $\Lambda$ based on the following constrained optimization problem:

$$\begin{aligned} \Lambda_{t+1} &= \arg \min_{\Lambda \in \mathbb{R}^n} \frac{1}{2} \left\| \Lambda - \Lambda_t \right\|_2^2 + C\xi^2 \\ &\text{subject to} \;\; g(\langle y_1, y_2 \rangle) - g(\langle \hat{y}_1, \hat{y}_2 \rangle) \geq 1 - \xi. \end{aligned} \tag{3.1}$$

The first term of the objective function, $\frac{1}{2} \left\| \Lambda - \Lambda_t \right\|_2^2$, says that $\Lambda$ should change as little as possible (in Euclidean distance) from its current value $\Lambda_t$. The constraint, $g(\langle y_1, y_2 \rangle) - g(\langle \hat{y}_1, \hat{y}_2 \rangle) \geq 1 - \xi$, says that the score of the correct task pair should be larger than the score of the incorrect task pair by at least $1 - \xi$. Ideally, $\xi = 0$, so that this enforces the condition that the margin (between correct and incorrect scores) should be 1.

The purpose of $\xi$ is to introduce some robustness to noise. We know that inevitably, the user will occasionally make a mistake in providing feedback. This could happen because of a slip in the UI or because the user is actually inconsistent about how resources are associated with tasks. In any case, the second term in the objective function, $C\xi^2$, serves to encourage $\xi$ to be small. The constant parameter $C$ controls the tradeoff between taking small steps (the first term) and fitting the data (driving $\xi$ to zero). Crammer, et al. [2006] show that this optimization problem has a closed-form solution, so it can be computed in time linear in the number of features and the number of classes.

The Passive-Aggressive algorithm is very attractive. However, one risk is that $\Lambda$ can still become large if the algorithm runs for a long time, and this could lead to overfitting. Hence, we modified the algorithm to include an additional regularization penalty on the size of $\Lambda$. The modified optimization problem is the following:

$$\begin{aligned} \Lambda_{t+1} &= \arg\min_{\Lambda\in\mathbb{R}^n} \frac{1}{2}\left\|\Lambda - \Lambda_t\right\|_2^2 + C\xi^2 + \frac{\alpha}{2}\left\|\Lambda\right\|_2^2 \\ &\text{subject to} \;\; g(\langle y_1, y_2\rangle) - g(\langle \hat{y}_1, \hat{y}_2\rangle) \geq 1 - \xi. \end{aligned} \tag{3.2}$$

The third term in the objective function, $\frac{\alpha}{2}\left\|\Lambda\right\|_2^2$, encourages $\Lambda$ to remain small. The amount of the penalty is controlled by another constant parameter, $\alpha$.

We now derive a closed-form update rule for this modified Passive-Aggressive algorithm. First, let us expand the $g$ terms. Define $\mathbf{Z}_t = \langle \mathbf{Z}_t^1, \mathbf{Z}_t^2, \mathbf{Z}_t^3 \rangle$ where

$$\begin{aligned} \mathbf{Z}_t^1 =& \mathbf{F}_A(\mathbf{X}_{t-1}, y_1) + \mathbf{F}_A(\mathbf{X}_t, y_2) \\ & - \mathbf{F}_A(\mathbf{X}_{t-1}, \hat{y}_1) - \mathbf{F}_A(\mathbf{X}_t, \hat{y}_2) \\ \mathbf{Z}_t^2 =& (\phi(y_1 \neq y_2) - \phi(\hat{y}_1 \neq \hat{y}_2))\, \mathbf{F}_S(\mathbf{X}_{t-1}) \\ \mathbf{Z}_t^3 =& (\phi(y_1 \neq y_2) - \phi(\hat{y}_1 \neq \hat{y}_2))\, \mathbf{F}_S(\mathbf{X}_t). \end{aligned}$$

Plugging this into the above optimization problem allows us to rewrite the inequality constraint as the following simple form:

$$\Lambda \cdot \mathbf{Z}_t \geq 1 - \xi.$$

We can then derive the following result:

**Lemma 1** *The optimization problem (3.2) has the closed-form solution*

$$\Lambda_{t+1} := \frac{1}{1+\alpha}(\Lambda_t + \tau_t \mathbf{Z}_t), \tag{3.3}$$

*where*

$$\tau_t = \frac{1 - \Lambda_t \cdot \mathbf{Z}_t + \alpha}{\|\mathbf{Z}_t\|_2^2 + \frac{1+\alpha}{2C}}. \tag{3.4}$$

A detailed proof is presented in the Appendix. The update rule (3.3) can be viewed as shrinking the current weight vector $\Lambda_T$ and then adding in the incorrectly-classified training example with a step size of $\tau_t/(1+\alpha)$. The step size is determined (in the numerator of (3.4)) by the size of the error $(1 - \Lambda_t \cdot \mathbf{Z}_t$

and (in the denominator) by the squared length of the feature vector and a correction term involving $\alpha$ and $C$.

The time to compute this update is linear in the number of features. Furthermore, the cost does not increase with the number of classes, because the update involves comparing only the predicted and correct classes.

It is worth asking how much accuracy is lost in order to obtain such an efficient online algorithm compared to a batch algorithm (or, more generally, compared to the best possible algorithm). By extending existing results from computational learning theory, we can provide a partial answer to this question. Let us assume that the length of each vector $\mathbf{Z}_t$ is no more than some fixed value $R$:

$$\|\mathbf{Z}_t\|_2 \leq R.$$

This is easily satisfied if every feature has a fixed range of values. Suppose there is some other algorithm that computes a better weight vector $\mathbf{u} \in \mathbb{R}^n$. Let $\ell_t = \max\{0, 1 - \Lambda_t \cdot \mathbf{Z}_t\}$ be the hinge loss of $\Lambda_t$ at time $t$. The hinge loss is the amount by which the constraint in Problem (3.2) fails to be satisfied. The hinge loss will be at least 1 for misclassified examples. It is between 0 and 1 for examples that are correctly classified by only a small margin less than 1. Similarly, let $\ell_t^* = \max\{0, 1 - \mathbf{u} \cdot \mathbf{Z}_t\}$ denote the hinge loss of $\mathbf{u}$ at iteration $t$. With these definitions, we can obtain a result that compares the accuracy of the online algorithm after $T$ examples to the total hinge loss of the weight vector $\mathbf{u}$ after $T$ examples:

**Theorem 1** *Assume that there exists a vector* $\mathbf{u} \in \mathbb{R}^n$ *which satisfies* $h = \frac{1-\alpha^2}{R^2+\frac{1+\alpha}{2C}} -$

$\frac{\alpha}{2+\alpha}\|\mathbf{u}\|_2^2 > 0$. *Given $\alpha < 1$, the number of prediction mistakes made by our algorithm is bounded by $m \leq \frac{1}{h}\|\mathbf{u}\|_2^2 + \frac{2C}{h(1+\alpha)}\sum_{t=0}^{T}(\ell_t^*)^2$.*

A detailed proof is presented in the Appendix. This theorem tells us that the number of mistakes made by our online algorithm is bounded by the sum of (a) the squared hinge loss of the ideal weight vector **u** multiplied by a constant term and (b) the squared Euclidean norm of **u** divided by $h$. This bound is not particularly tight, but it does suggest that as long as **u** is not too large, the online algorithm will only make a constant factor more errors than the ideal (batch) weight vector.

Theoretical analysis can provide one more insight into the behavior of the online algorithm. Because of the penalty $\frac{\alpha}{2}\|\Lambda\|^2$ on the weight vector, it turns out that this algorithm automatically decreases the influence of older training instances. Let $I_t$ be the set of instances $\mathbf{Z}_i$ such that $i < t$ and $\mathbf{Z}_i$ caused an update. From Lemma 1, the learned score function at iteration $t$ can be rewritten as

$$g_t(\mathbf{Z}) = \text{sign}\left(\sum_{i \in I_t} \frac{\tau_i}{(1+\alpha)^{|I_t \setminus I_i|}}\mathbf{Z}_i \cdot \mathbf{Z}\right) \tag{3.5}$$

where $|I_t \setminus I_i|$ is the number of elements in the set difference of $I_t$ and $I_i$. This indicates that the algorithm shrinks the influence of the old observations and puts more weight on the recent ones.

### 3.3.3 Pragmatic Issues in Selecting Training Examples

If the user were behaving correctly all the time, then every time the user declared a task switch, we would obtain a training example of a correct switch, and whenever the user did not declare a task switch, we would get an example of a non-switch. But of course the whole reason to include an intelligent switch predictor is because the user makes mistakes. When can we reliably train the classifier? We answer this question as follows. We create positive examples of task switches whenever (a) the user declares a task switch or (b) the user confirms a predicted switch (possibly after modifying the predicted task and/or the predicted switch point). We create negative examples of task switches for (a) the first $d = 10$ minutes after a declared or accepted task switch and (b) the first $d = 10$ minutes after the user Removes a predicted switch. The rationale for the 10-minute interval is that the declared task is usually correct immediately after the user switches to the task. But as the amount of time since the confirmed switch increases, it becomes more and more likely that the user has switched without making an explicit declaration. Using a fixed 10-minute window can be viewed as a crude Bayesian prior over the likelihood of the user forgetting to declare a switch. An interesting direction for future research would be to learn such a model based on the actual behavior of the user.

There is an additional pragmatic issue that results from the combination of affordances provided by the UI. In particular, consider the "dragging" affordance. Consider Figure 3.2a, where a prediction has been made immediately after five Firefox web pages and just prior to two Word documents. Suppose that this prediction was incorrect in that there was no task switch. Then the user would right-click and select “Remove” from the pop-up menu. Our current system treats this as a negative training example, and this sets up the start of a 10-minute segment in which all user behavior will be trusted and converted to training examples.

However, we can infer more from the user’s Remove action. Suppose that there had been a task switch at an earlier point (e.g., immediately after the first two Firefox web pages). In that case, the UI would have permitted the user to drag the red triangle to that point and then confirm. Therefore, under the assumption that the user knows about this affordance, we can infer from the fact that the user did not do this, that there was *not* a task switch anywhere in the time interval between the first Firefox page and the last Word file. Our implementation does not currently do this, but it raises an interesting question for future research. We only discovered this issue after the UI was deployed. It would be interesting to develop a design tool that could discover these kinds of counter-factual inferences earlier in the design process.

## 3.4 Experimental Evaluation

We deployed TaskTracer on Windows machines in our research group and collected data from two regular users, both of whom were fairly careful about declaring switches. In addition, an SVM-based version of TaskPredictor2 was running throughout this time, and the users tried to provide feedback to the system

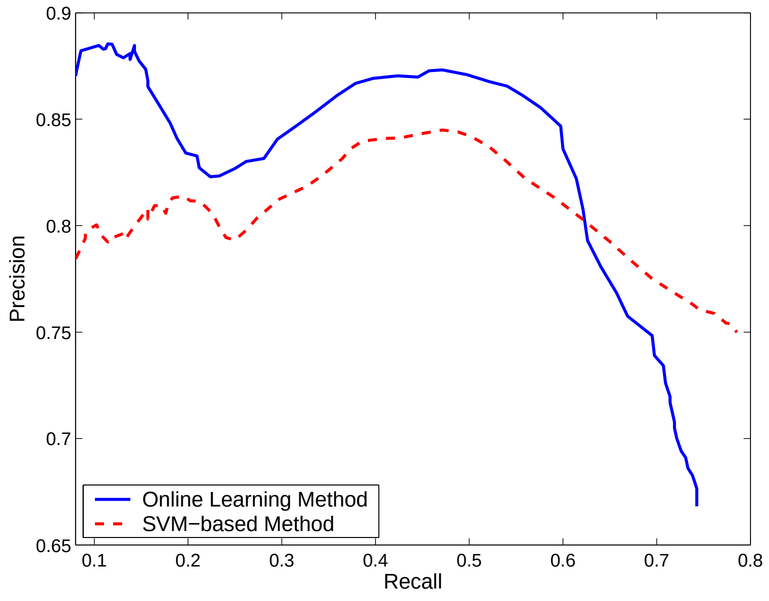


Figure 3.3: User 1: Precision of different learning methods as a function of the recall, created by varying the threshold.

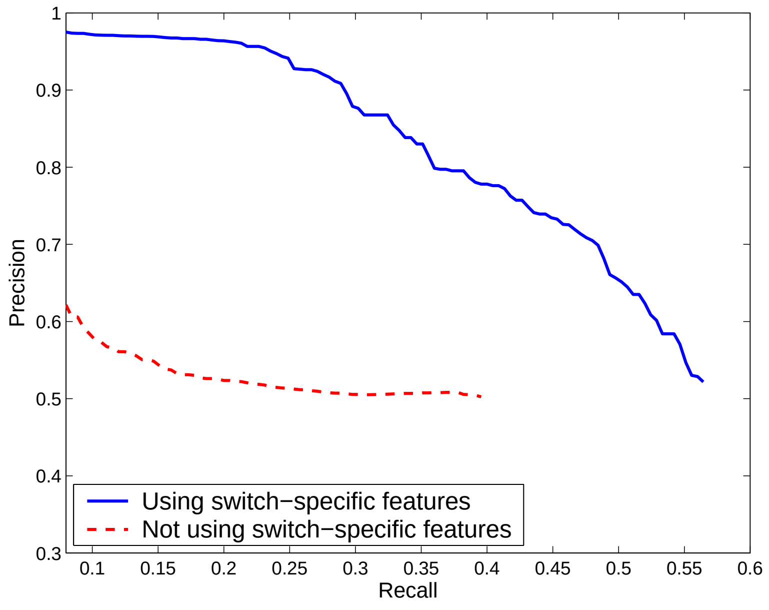


Figure 3.4: User 2: Precision of different learning methods as a function of the recall, created by varying the threshold.

throughout.

The first user (User 1) is a "power user", and this dataset records 4 months of daily work, which involved 299 distinct tasks, 65,049 instances (i.e., information vectors), and 3,657 task switches. The second user (User 2) ran the system for 6 days, which involved 5 tasks, 3,641 instances, and 359 task switches.

To evaluate the online learning algorithm, we make the assumption that these task switches are all correct, and we perform the following simulation. Suppose the user forgets to declare every fourth switch. We feed the information vectors to the online algorithm and ask it to make predictions. A switch prediction is treated as correct if the predicted task is correct and if the predicted time of the switch is within 5 minutes of the real switch point. When a prediction is made, our simulation provides the correct time and task as feedback.

The algorithm parameters were set based on experiments with non-TaskTracer benchmark sets. We set $C = 10$ (which is a value widely-used in the literature) and $\alpha = 0.001$ (which gave good results on the benchmark data sets).

Performance is measured by *precision* and *recall*. Precision is the number of switches correctly predicted divided by the total number of switch predictions, and Recall is the number of switches correctly predicted divided by the total number of undeclared switches. We obtain different precision and recall values by varying the score confidence threshold required to make a prediction.

The results comparing our online learning approach with our previous approach described in Section 2.3 are plotted in Figures 3.3 and 3.4. Our previous approach only uses the bag of words from the window titles and pathname/URL to do

Table 3.1: Usage Statistics for User 1; SVM Version

| | | | Confirmed | | | |
|---|---|---|---|---|---|---|
| Switch Type | Ignored | Removed | NoChange | ChangeTask | ChangeTime | ChangeBoth |
| Low Confidence | 1 | 1 | 1 | 1 | 2 | 0 |
| Medium Confidence | 10 | 2 | 4 | 0 | 3 | 0 |
| High Confidence | 31 | 33 | 32 | 2 | 14 | 1 |
| User-Declared | N/A | 5 | N/A | 2 | 21 | 1 |

inference. The new approach incorporates much richer contextual information to detect task switches. This makes our new approach more accurate. Compared with the previous approach which must train multiple binary SVMs, the online learning approach is also much more efficient. On an ordinary PC, it only took 4 minutes to make predictions for User 1's 65049 instances while the previous approach needed more than 12 hours!

### 3.4.1 Qualitative Evaluation and the User Experience

Within our group, we are currently using the SVM-based version of the system. It is exactly the same as the system described in this chapter, except that it employs the LibSVM implementation [Chang and Lin, 2001] to train the classifier. Table 3.1 shows the actual results of the four-month usage period for User 1. A total of 138 predictions were made. Of these, 42 (30.4%) were ignored (probably because the user didn't notice them when the UI was buried under another window). Of the 96 non-ignored predictions, the user removed 36 (37.5%) of them. The user confirmed 37 (38.6%) without change, 3 more (3.1%) after changing the task, 19 (19.8%) after changing the time, and 1 more (1.0%) after changing both the time and the task.

Qualitatively, User 1 reported that TaskPredictor2 was very accurate. When it makes a prediction, the prediction is usually either correct or sensible, by which we mean that the predicted task is one that is related to the correct task. For example, a challenging case for task prediction arises when the user is working on task A2 and accesses a resource that was previously only known to be associated with task A1. Two common scenarios where this arises are (a) using a document as a template for a new task (e.g., opening up an old syllabus, editing it to change the course name, meeting time, and so on, and then saving it) and (b) assembling summary documents (project reports, curriculum vitae, annual performance reviews) by copying information from multiple source documents. In such cases, TaskPredictor2 will predict that the user has switched to task A1. This is a sensible prediction, and users are not surprised by these kinds of predictions. However, from the user's point of view, he/she is working on task A2, so the user typically Removes or ignores such predictions. This causes the source documents to become associated with A2. It isn't clear whether this result is good or bad. A good aspect of this is that at some later time, when the user returns to working on task A2, TaskTracer will include the source documents in the list of resources associated with A2. This can provide an answer to the question "Where did I get those slides from?". A bad aspect of this is that when the user accesses the source documents at some later time, TaskPredictor2 may predict that the user is switching to task

A2 when the user is really switching to task A1. We have attempted to deal with this case by including the “strength of association” feature in the information vector. Ideally, if a resource is more strongly associated with A1 than with A2, then this will cause the predictor to predict a switch to A1 in this case.

## 3.5 Conclusion

This chapter has described TaskPredictor2, a novel task-switch predictor for the desktop environment. Its user interface displays task switches [Smith *et al.*, 2003; Czerwinski *et al.*, 2004] on a time line that shows accessed resources. This gives the user contextual hints and makes it easy for the user to adjust the time at which a task switch (declared or predicted) actually occurred. Instead of simply reminding the user about the switch, it allows the user to edit the task switch history and thereby adjust resource-task associations. The learning component exploits rich contextual information, including both task-specific features and switch-specific features, to predict task switches. The predictor parameters are trained via a novel online learning algorithm. This algorithm is more accurate, more efficient in memory, and runs much faster than our previous SVM-based method.

# Chapter 4 – Frequent Workflow Model Mining for Workflow Recognition

In order to help the user organize and re-find information, TaskTracer requires that the user explicitly names each of his/her activities, which imposes additional overhead and is more appropriate for larger chunks of activity such as long-lived projects. Such an approach lacks an understanding of the *state* of the activity, so it can't tell whether an activity is completed or ongoing, nor can it provide useful reminders to the user about actions that still need to be performed to complete the activity. Such activities correspond to "tasks" – the highest level of activities in our definition as described in Section 1.3.

In this chapter, we address the above issues by extending "activities" to workflows [Shen *et al.*, 2009a]. Workflows can be formally prescribed procedures (e.g., submitting requests for travel authorization) or informal, idiosyncratic procedures (e.g., keeping track of fantasy football results). This chapter describes an algorithm for mining frequent workflow models and then describes how to convert them into Logical Hidden Markov Models for workflow recognition [Kersting *et al.*, 2006].

Discovering desktop workflows is difficult because they unfold over extended periods of time (days or weeks) and they are interleaved with many other workflows because of user multi-tasking. This chapter describes an approach to discovering desktop workflows based on rich instrumentation of information flow actions such as copy/paste, SaveAs, file copy, attach file to email message, and save attachment. These actions allow us to construct a graph whose nodes are files, email messages, and web pages and whose edges are these information flow actions. A class of workflows that we call *work procedures* can be discovered by applying graph mining algorithms to find frequent subgraphs. They are then converted into Logical Hidden Markov Models for workflow recognition.

## 4.1 Overview

Knowledge workers frequently change activities, either by choice or through interruptions [Mark *et al.*, 2005]. With an increased number of activities and activity switches, it becomes more and more difficult for knowledge workers to keep track of the state of each of their activities. Many attempts have been made to develop effective To-Do managers, but these impose additional overhead on desktop work and they do not capture many of the user's activities [Bellotti *et al.*, 2003]. One way to reduce this overhead and increase the coverage of a To-Do manager would be if the computer could automatically recognize which activities were being executed and track their status in an intelligent To-Do manager.

In our work on the TaskTracer system [Dragunov *et al.*, 2005; Shen *et al.*, 2007], we took an initial step in this direction by applying machine learning methods to predict the current activity of the user based on the resources (files, web pages, email messages) that the user accesses. This approach requires that the user explicitly name each of his/her activities, which imposes additional overhead and is more appropriate for larger chunks of activity such as long-lived projects. Such activities are called “tasks” in our system. An additional shortcoming of this approach is that it lacks an understanding of the *state* of the task, so it can’t tell whether a task is completed or ongoing, nor can it provide useful reminders to the user about actions that still need to be performed to complete the task. So while the TaskTracer approach is very valuable for helping the user recover from interruptions of long-duration activities, it does not provide any support for managing To-Do lists.

A significant portion of the desktop activity of knowledge workers involves executing workflow procedures. These can be formally prescribed procedures (e.g., submitting requests for travel authorization) or informal, idiosyncratic procedures (e.g., keeping track of fantasy football results). Suppose that the computer had models of these workflows. With such models, it would be able to detect when a new workflow instance was initiated, track the current state of execution of each workflow instance, and determine when the execution was complete. This would allow it to automatically populate and maintain an intelligent To-Do list for the knowledge worker.

If the workflow models were sufficiently precise, they could also provide the basis for partial automation of workflows. For example, consider the simple workflow “provide comments on a document”. In this workflow, a document arrives as an email attachment. The user saves the document into an appropriate folder, edits it (optionally saving it under a new name), and then attaches it to a reply email. A smart desktop assistant might be able to recognize the start of the workflow by analyzing the initial email message. In which case, it could place an entry on the To-Do list. When the user was ready to work on this action item, the assistant could provide a one-click way of opening the email message, saving the attachment, and opening the file. Later, after the file had been edited and saved, it could provide one-click automation for opening the email message, initiating a reply, and attaching the file to the reply message. These simple automation steps could avoid the need for the user to find the old email message, remember where the file was saved, and so on. We hypothesize that such support could avoid the need to resort to desktop search tools to remember and re-access such "lost" items.

While this vision of an intelligent workflow assistant is very attractive, it has one critical flaw: how can the computer acquire detailed models of desktop workflows? Some researchers have addressed this problem by providing easy ways for users to define the workflow procedures themselves [Leshed *et al.*, 2008]. We propose instead to automatically discover the workflows by observing the user's desktop activity and detecting repeated sequences of actions. This is very challenging for a number of reasons. First, these workflows take place over extended periods of time (days, weeks, months). Second, they are interleaved with thousands of irrelevant actions that either do not belong to any workflow or belong to other workflows. Indeed, for a workflow such as "Review conference paper", many instances of that workflow will be executing simultaneously. How can we detect and untangle these interleaved instances?

We present an approach based on capturing information flow actions (also called provenance links [Muniswamy-Reddy *et al.*, 2006; Shah *et al.*, 2007; Frew *et al.*, 2008]). We define a *work procedure* as a directed graph whose nodes are resources (files, email messages, web pages) and whose arcs are actions such as SaveAs, copy/paste, save email attachment, attach file to email, upload file to web page, download file from web page, and so on. With sufficient instrumentation of desktop applications, we can capture the complete set of these provenance links. This allows us to capture the user's desktop behavior as a large graph of resources and provenance links. Each work procedure instance is a connected subgraph in this graph. To discover these instances, we extend an existing subgraph mining algorithm to find frequently occurring subgraphs.

The information flow graph solves the problem of temporally-extended interleaved work procedures. The provenance links in the graph are the same regardless of whether the work procedure was executed in a single day or over many months. And the topology of the graph does not depend on how multiple work procedure instances were interleaved. This leads to a powerful savings in computation. Consider two work procedure instances each involving 8 actions. There exist $\binom{16}{8} = 12,870$ possible interleavings of these actions, but there is only one information flow graph.

To acquire the information flow graph, we employed the TaskTracer system [Dragunov *et al.*, 2005; Shen *et al.*, 2007]. TaskTracer already tracks every resource (file, folder, web page, email message, email contact) accessed by the user, so these provide the nodes in the graph. To acquire the provenance links, we extended TaskTracer to capture several kinds of provenance links (File copy, File rename, Copy/Paste, SaveAs, Save Email Attachment, Attach to Email, Download from web page, Upload to web page). This does not capture all of the relevant provenance links, so in this chapter we performed additional automatic analysis of the user's resources to identify additional links as described below.

The mined work procedures are then converted into logical Hidden Markove Models [Kersting *et al.*, 2006]. Each edge corresponds to a state in the logical Hidden Markov Model. We also introduce a "background" state between each pair of work procedure states. Such "background" states correspond to the possible cases that the user switches to other activities after executing some workflow steps.

The remainder of the chapter is structured as follows. The next section reviews related work. Then, we motivate the work procedure discovery problem with some typical work procedure cases and discuss its potential usage. Next, we provide a brief overview of the TaskTracer system used for our research and present a user interface for displaying the provenance information. We describe our methods for building the information graph, for finding frequent generalized closed patterns, for assigning appropriate action sequences and for converting the mined models into logical Hidden Markov Models. The last section presents experimental results on the accuracy of the approach.

## 4.2 Related Work

Several other researchers have worked on problems similar to the one we address in this chapter.

**Business workflow mining**. Traditionally, a *workflow* is defined as a partial or total automation of a business process in which a collection of activities must be executed by humans or machines according to certain procedural rules [Agrawal *et al.*, 1998; Medeiros *et al.*, 2003; Greco *et al.*, 2005; Ellis *et al.*, 2006]. Workflow management systems help to execute, monitor and manage work process flow and execution. They (partially) automate the definition, creation, execution, and management of work processes through the use of software. Their transaction logs record information of each executed process. The activity of using computer software to examine theses records and derive various structural data results is called workflow mining [Ellis *et al.*, 2006].

Our work procedure discovery problem can be thought of as a special case of the workflow mining problem with two differences: (a) the traditional workflow mining problem tries to find a model expressing the business process of an organization and the execution of a model usually involves multiple people, while we focus on individual knowledge workers and the discovered model is executed by a single user; (b) in the traditional workflow mining problem, the instances of the workflows are assumed to already be identified, whereas in our work we must discover the instances first.

**Motif discovery**. Another related line of work is motif discovery in biological sequence data and continuous time-series data [Lin *et al.*, 2002; Chiu *et al.*, 2003; Tanaka *et al.*, 2005; Hamid *et al.*, 2005; Minnen *et al.*, 2007a; Minnen *et al.*, 2007b]. A *motif* is a frequently-appearing pattern. A typical motif discovery approach first applies a sliding window to divide the temporal data into a set of subsequences, and then builds a similarity matrix among these subsequences. This matrix is then employed to select seed motif occurrences and locate additional motif occurrences. Motif discovery is useful for various time-series data mining tasks such as mining association rules in time-series data, time series classification, anomaly/interestingness detection, and so on [Chiu *et al.*, 2003].

A motif occurrence corresponds to a subsequence of the time-series data. Every event point between the start point and the end point belongs to that motif occurrence. This is very different from the work procedures that we seek to discover, which are interleaved and intermixed with unrelated (irrelevant) actions.

**Desktop activity modeling**. To build intelligent personal assistants to help users organize their work, people have tried to automatically extract understandable descriptions of a desktop user's activities [El-Ramly *et al.*, 2002; Mitchell *et al.*, 2006; Amershi and Conati, 2007; Fern *et al.*, 2007b]. Such descriptions can help researchers understand users' behavior and design smarter intelligent agents. However, unlike our work procedures, these descriptions do not support (semi-) automated execution of the discovered activities. Instead, each activity is usually represented as a set of objects (people, documents) or as a cluster of action sequences. Many of these approaches assume prior knowledge about the number of activities and their duration.

**Email activity management**. Email activity management systems help knowledge workers manage their daily work life by tracking email exchanges. The work by Bellotti, et al. [2003], supports the manual population of an activity template from emails. Dredze, et al. [2006] classify email messages into activities by analyzing their content. The work of Kushmerick, et al. [2005] is quite similar to our work in that they also attempt to discover workflows and track the status of those workflows. However, they focus on e-commerce transactions where the email messages involve unique identifiers (e.g., order numbers; ebay item numbers), which simplifies the problem.

## 4.3 The Work Procedure Discovery Problem

### 4.3.1 Definitions

We now formally define the work procedure discovery problem.

**Definition 1** *A resource is a data object such as a file, email message, or web page.*

In our modified TaskTracer system, each resource is assigned a unique identifier the first time it is accessed. There are several subtleties involved in handling ephemeral resources such as email messages that are currently being composed or MS Office documents that have not yet been saved ("Document1", "Presentation2").

**Definition 2** *An information flow action (also called a provenance link) is a primitive or composite desktop action that directly links two resources.*

We directly instrumented the following information flow actions:

- Copy/paste. This involves instrumenting both the Copy event (to capture the resource from which the information is copied) and the Paste event (to capture the resource into which the information is pasted).
- Attach file to email. In order to capture the name of the file on disk, we added our own button to the MS Outlook user interface.
- Save email attachment. In order to capture the name of the saved file on disk, we added our own button to the MS Outlook user interface.
- Download file from web page. We wrote our own download manager as a plugin to Internet Explorer.
- Upload file to web page.
- Copy or Rename file in Windows Explorer.

**Definition 3** *An* information flow graph *is a labeled directed graph where the nodes are resources and the arcs are information flow actions. If there are multiple information flow actions linking two nodes, the graph contains a single arc labeled with the set of corresponding information flow actions.*

The information flow graph is the raw graph of resources and actions constructed from observing the user's desktop activity. Note that there may be multiple actions connecting two resources. For example, suppose the user performs a SaveAs and then accidentally deletes part of a document. He might copy/paste that part from the previous version. Another simple example is that there are often multiple copy/paste actions from one resource to another.

After the basic information flow graph is constructed, it is augmented with additional arcs representing chain relations:

**Definition 4** *A* resource-action chain *is a sequence of resources joined by actions of a single type.*

For example, consider a chain of files created by a sequence of SaveAs actions: F1 was saved as F2 which was saved as F3 which was saved as F4. We can collapse this into a single step F1 $\rightarrow$ F4 for action "SaveAs…", where "SaveAs…" is a resource-chain action. A slightly more general example is a chain of email messages: user A sends message M1 to user B, user B replies with message M2 to user A, and then user A replies with message M3 to user B. We can collapse this into a single step M1 $\rightarrow \ldots$ M3 for the resource chain action "EmailSend…". More formally, for the actions "SaveAs", "EmailSend", and "EmailReceive", we define additional resource chain actions "SaveAs…", "EmailSend…", and "EmailReceive…". Two resources linked by one of these special actions denote a resource-action chain linked by the appropriate primitive actions.

**Definition 5** *A* work procedure $\mathcal{WP} = \langle \mathcal{R}, \mathcal{A} \rangle$ *is a labeled directed graph where* $\mathcal{R}$ *is the set of resources involved in* $\mathcal{WP}$ *and* $\mathcal{A}$ *is the set of actions (including resource-chain actions) performed in* $\mathcal{WP}$*. Each resource* $r \in \mathcal{R}$ *corresponds to a node. Each action* $a \in \mathcal{A}$ *corresponds to a directed edge in the graph, with the target resource as the end node. The label of a node is the resource type of that node* *(document, web page) and the label of an edge is the type of the action (copy/paste, attach file).*

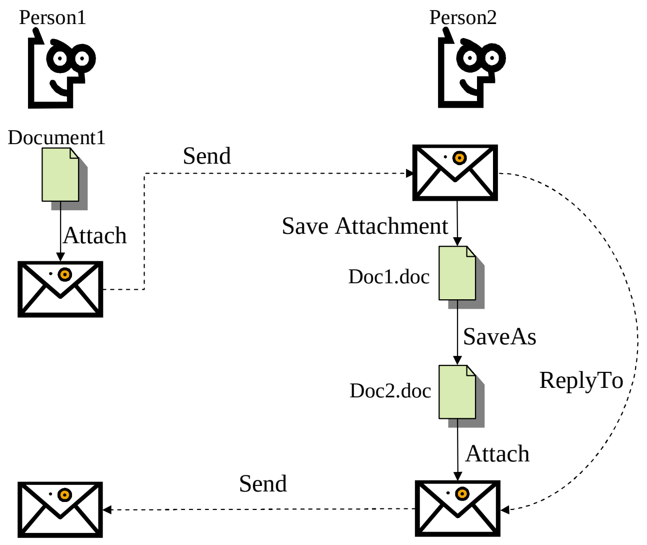
A work procedure is an abstract graph in which the specific identity of the resources and information flow actions is replaced only by their types. Unlike in the information flow graph, two nodes in a work procedure may only be joined by a single edge.

Note that a work procedure typically does not contain all of the information needed to automate the procedure. For example, editing a document involves more than just performing a SaveAs, and uploading a file typically also involves logging in to a web page, filling out a form, and so on.

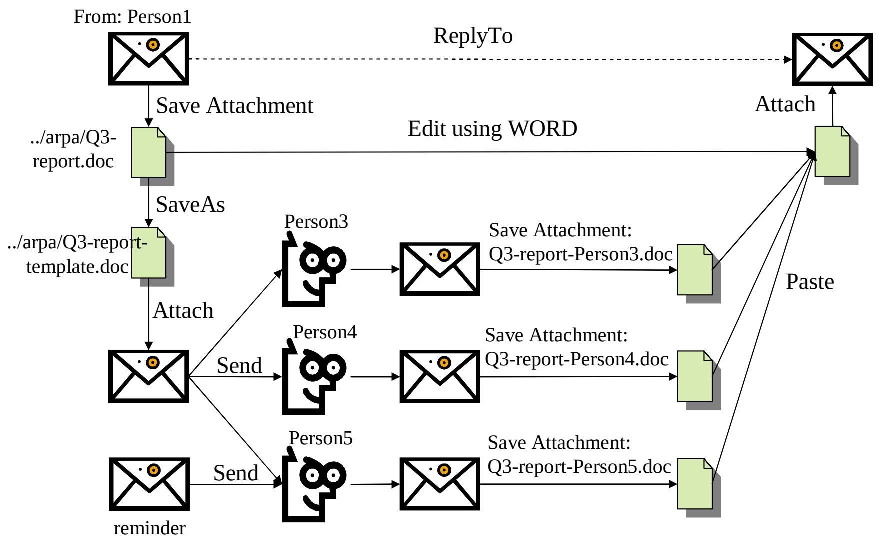
**Definition 6** *A* work procedure instance *consists of a pair of a work procedure and a subgraph of an information-flow graph joined by a mapping that maps each node and edge in the work procedure into a node or edge of the same type in the information-flow graph. If an edge is labeled with a resource-chain action, then it can be mapped to a chain of 0 or more resources in the information-flow graph.*

Given an information-flow graph and a work procedure, there may be several different work procedure instances in the information-flow graph. These instances may even share nodes and edges.

**Definition 7** *The* work procedure discovery problem *is the following: Given an information-flow graph, discover a set of work procedures to cover as much of the graph as possible and that will generalize to additional information-flow graphs.*



(a) Comment on a document


From: Person1
ReplyTo
Save Attachment
Attach
Edit using WORD
../arpa/Q3-report.doc
SaveAs
../arpa/Q3-report-template.doc
Person3
Save Attachment: Q3-report-Person3.doc
Paste
Attach
Person4
Send
Save Attachment: Q3-report-Person4.doc
Person5
Send
Save Attachment: Q3-report-Person5.doc
reminder

(b) Prepare quarterly report

Figure 4.1: Two typical work procedure examples.

### 4.3.2 Examples of Work Procedures

A simple work procedure *Provide Comments on a Document* is presented in Figure 4.1(a). Person2 receives an email message from Person1 requesting comments on an attached Word file. Person2 saves the attachment in a folder associated with the relevant project. He opens the document and uses SaveAs to save it with a new name. After editing the file, Person2 replies to the original email message and attaches the edited file. If the system can recognize instances of this work procedure, it can automatically create a To-Do item when the email arrives and remind Person2 about the progress. After detecting that Person2 has finished editing the document, it can offer to send it back to Person1.

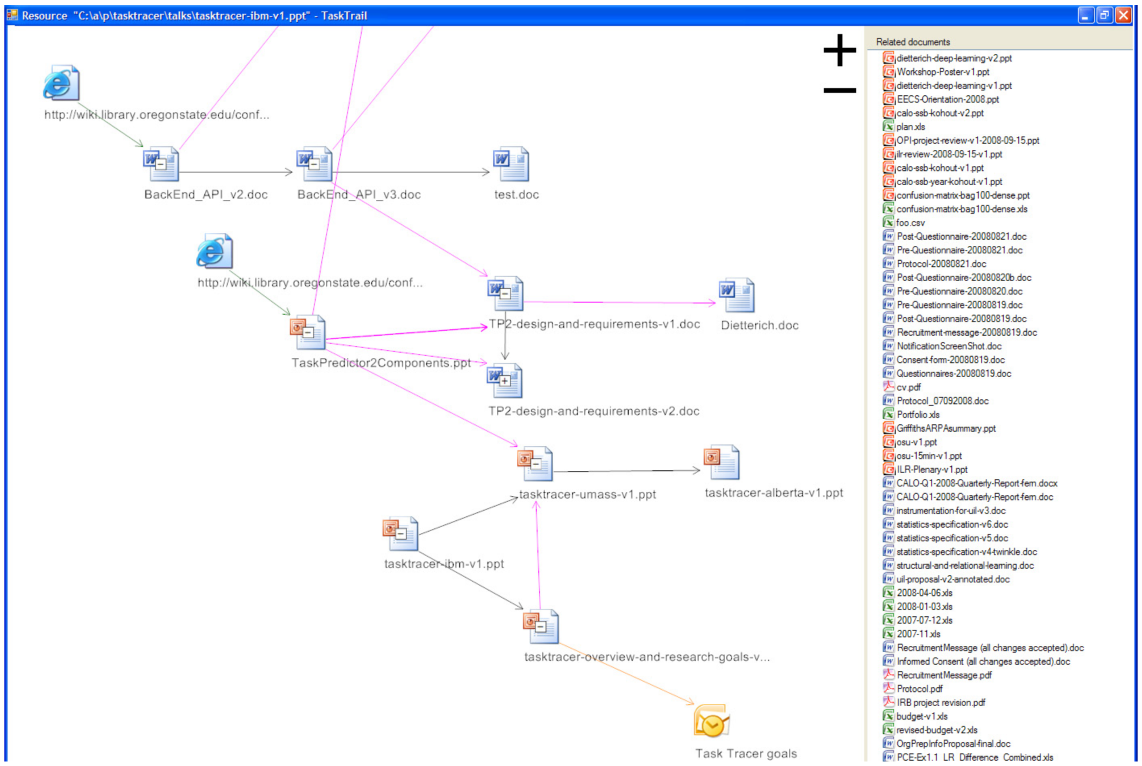
A more complex work procedure *Prepare Quarterly Report* is presented in Figure 4.1(b). Person2 receives an email from Person1 requesting the quarterly report with an attached Word file and possibly a deadline. Person2 saves the attachment into the relevant folder. He opens the file and does some editing. Let us suppose that Person2 needs to obtain information from three additional people (Person3, Person4, and Person5) to complete the report. He composes an email message to these people requesting their contributions and attaching this file as the template. As replies arrive, he saves their attached files in the same folder. As the deadline approaches, he sends a reminder to those people who have not yet sent in their contributions. When the deadline arrives, he opens the original file and all of the various contributions, and he copies/pastes their materials into the file. Finally he replies to the original email from Person1 with the attached report file. If the system could recognize instances of this work procedure, it could automatically create a To-Do item when the email arrives. Later, it could save the attachment and open it in Word. On request, it could create an outgoing email message and attach the template. As the replies come in, it could track them and save the attachments in the right folder. It could also offer to send reminders to the people who have not yet responded. It could also offer to open the template file and all of the contribution files to help the user edit the report. Finally, it could offer to compose a reply email to Person1 and attach the final version of the file.

In this chapter, we are interested in work procedures that can provide benefit to the user either through automatic creation and state tracking or through (partial) automation. Hence, we make the following assumptions:

- A work procedure is a (weakly) connected graph. We assume that we can observe (or reconstruct) enough information flow actions so that each work procedure is a weakly connected graph. This means that every node can be reached from every other node if we ignore the directions of the information flow links.

- A work procedure involves at least $k = 3$ resources. Automatic tracking of work procedures is likely to be more useful if there are many resources involved, because users have more difficulty keeping track of the state of more complex procedures.

- A work procedure involves resources other than emails. A procedure only consisting of emails is not our focus. First, there are already very good email


Resource "C:\a\p\tasktracer\talks\tasktracer-ibm-v1.ppt" - TaskTrail
http://wiki.library.oregonstate.edu/conf...
BackEnd_API_v2.doc
BackEnd_API_v3.doc
test.doc
http://wiki.library.oregonstate.edu/conf...
TP2-design-and-requirements-v1.doc
Dietterich.doc
TaskPredictor2Components.ppt
TP2-design-and-requirements-v2.doc
tasktracer-umass-v1.ppt
tasktracer-alberta-v1.ppt
tasktracer-ibm-v1.ppt
tasktracer-overview-and-research-goals-v...
Task Tracer goals

Related documents
- dietterich-deep-learning-v2.ppt
- Workshop-Poster-v1.ppt
- dietterich-deep-learning-v1.ppt
- EECS-Orientation-2008.ppt
- calo-ssb-kohout-v2.ppt
- plan.xls
- OPI-project-review-v1-2008-09-15.ppt
- ilr-review-2008-09-15-v1.ppt
- calo-ssb-kohout-v1.ppt
- calo-ssb-year-kohout-v1.ppt
- confusion-matrix-bag100-dense.ppt
- confusion-matrix-bag100-dense.xls
- foo.csv
- Post-Questionnaire-20080821.doc
- Pre-Questionnaire-20080821.doc
- Protocol-20080821.doc
- Post-Questionnaire-20080820b.doc
- Pre-Questionnaire-20080820.doc
- Pre-Questionnaire-20080819.doc
- Post-Questionnaire-20080819.doc
- Recruitment-message-20080819.doc
- NotificationScreenShot.doc
- Consent-form-20080819.doc
- Questionnaires-20080819.doc
- cv.pdf
- Protocol_07092008.doc
- Portfolio.xls
- GriffithsARPAsummary.ppt
- osu-v1.ppt
- osu-15min-v1.ppt
- ILR-Plenary-v1.ppt
- CALO-Q1-2008-Quarterly-Report-fem.docx
- CALO-Q1-2008-Quarterly-Report-fem.doc
- instrumentation-for-uil-v3.doc
- statistics-specification-v6.doc
- statistics-specification-v5.doc
- statistics-specification-v4-twinkle.doc
- structural-and-relational-learning.doc
- uil-proposal-v2-annotated.doc
- 2008-04-06.xls
- 2008-01-03.xls
- 2007-07-12.xls
- 2007-11.xls
- RecruitmentMessage (all changes accepted).doc
- Informed Consent (all changes accepted).doc
- RecruitmentMessage.pdf
- Protocol.pdf
- IRB project revision.pdf
- budget-v1.xls
- revised-budget-v2.xls
- OrgPrepInfoProposal-final.doc
- PCE-Ex1.1 LR Difference Combined.xls

Figure 4.2: Screenshot of TaskTrail displaying a real-case provenance connection.

threading tools that can help users track their work status [Lam *et al.*, 2002]. We believe it is also more difficult for the user to track work procedures involving different kinds of resources, because this requires working with multiple application programs.

## 4.4 Building the Information Flow Graph

We build the information flow graph by utilizing the provenance information collected by TaskTracer and augmenting it with links inferred by analyzing the contents of resources.

### 4.4.1 The TaskTracer System and Provenance Information

User's activities applied to resources are captured by the TaskTracer system [Dragunov *et al.*, 2005]. TaskTracer is an intelligent activity management system that allows users to organize and retrieve information based on their activities. TaskTracer collects various time-stamped user interactions (such as file new, open, save, text selection, copy/paste, windows focus, web navigation, email read/send, and so on) in Microsoft Office (Word, Excel, PowerPoint, Outlook), text and pdf files, Internet Explorer, and the Windows operating system. Each interaction generates events which are stored in a database. TaskTracer associates with each activity the set of resources accessed when performing that activity. It leverages this data and machine learning technology [Shen *et al.*, 2007] to configure the desktop to assist users in organizing and re-finding information.

Recent research [Blanc-Brude and Scapin, 2007] has shown that common search criteria, such as file modification time and even the document title, are usually remembered inaccurately. There are other resource attributes that are more easily remembered by users and could be helpful in re-finding documents. One such attribute is a resource's relationship to other resources [Gonçalves and Jorge, 2004; Blanc-Brude and Scapin, 2007]. Inspired by this and other work, we extended TaskTracer to capture a variety of provenance actions. These are also published as events within TaskTracer and recorded in the event database and in a provenance database. We have found that even without any further analysis, provenance data can provide answers to user queries that cannot be obtained in any other way. For example, a user might want to know such things as "Did I ever email this Word file to Bob?", "Did I save this email attachment, and if so, where did I put it?", or "What Powerpoint presentations did I copy slides from to produce this presentation?" We have modified the Windows and Outlook user interfaces so that the user can right-click on a file or email message and request a "provenance graph" to be displayed. *TaskTrail* is the TaskTracer component that displays these graphs and allows users to interact with them.

A real case using TaskTrail is shown in Figure 4.2. The user could not find the PowerPoint document he prepared for Alberta. But he remembered that this document was created from a previous presentation "tasktracer-ibm-v1.ppt" and he knew where that presentation was located. So he right-clicked "tasktracer-ibm-v1.ppt" and chose "Show TaskTrail" from the context menu. He instantly found what he was searching in the provenance graph. The left panel of TaskTrail visualizes the entry point resource and resources related via provenance (moving from left to right). The right panel lists all resources contained in the current provenance relationship. The user can change the display resolution, hover the mouse over nodes and links to get detailed information, and double-click on nodes to open resources.

### 4.4.2 Inferring Implicit Links

Some important provenance relations are not currently captured by TaskTracer because the corresponding instrumentation has not been implemented. Hence, for the data employed in this study, we analyzed the contents of various resources to reconstruct additional information flow relationships:

- **Email threading**: if email message $X$ is a reply to or forward of message $Y$, we want to create a "reply/forward" link from $Y$ to $X$. An email thread is a group of messages that are related by such links. By capturing email threads, we can connect many resources that seemed unrelated before. Those threading relationships can be accurately recovered by analyzing the subject lines and the RFC-822 header fields of email messages (such as *Message-ID*, *In-Reply-To*, *References* field, and so on).
- **Document conversion**: many users prefer to convert doc files into pdf files in order to send them as email attachments. We would like to create a "PDF-Print" link from this doc file to this pdf file. To capture such connection, we extract the text contents of each pdf file and compare it to the contents of doc files. If it matches a doc file, we create a link from this doc file to the pdf file.
- **URL click-through**: instead of attaching a file, the user sometimes provides a URL in the email message and asks the recipient to download the file. If the user clicks on a URL in an email and opens a website or downloads a file,

we wish to create a "Click-through/SaveAttachment" link from this email to the website or the file. For each navigation or download event, we check if the previous window in focus is the email client. If it is and the opened email message contains that URL, we create a "Click-through/SaveAttachment" link.

- **Email reference**: some email messages contain information about specific documents. For example, the user might receive an email regarding his paper submission and asking for a response. There should be a "Reference" link from this document to the email message. Such email usually contains the paper title in the email body. We use a pre-trained key value extractor [Cohen, 2004] and "false positive" regular expressions [Lam *et al.*, 2002] to extract key words such as document titles in email messages. We define the first 200 words of a document to be the abstract of that document. If any key word extracted from the email message appears in the abstract, there is a potential connection. If an email matches multiple documents (this could happen when the user edits and saves the document with several versions), we only create one link from the document which has the closest time distance to the email.

## 4.5 Mining Frequent Work Procedures

Given information flow graphs of desktop users, we can infer work procedures by identifying recurring patterns within these graphs. For computational reasons, we do this in two steps. First, we ignore the labels on the links in the graph and search for frequently-occurring subgraphs. Then, we choose action types for each edge in the discovered subgraphs by analyzing all instances of those subgraphs to choose the most frequently-occurring combinations of action types.

### 4.5.1 Mining Closed Frequent Relations

Our goal is to find closed weakly-connected patterns in the information flow graph that are frequent (i.e., have at least $s$ instances). A directed graph $G$ is weakly connected if replacing all of its directed edges with undirected edges will produce a connected (undirected) graph. The frequency of a pattern graph $P$ is the number of instances of $P$ in the information flow graph $G$. A pattern graph $P$ is frequent if its frequency is greater than or equal to a specified minimum support threshold $s$. A frequent pattern $P$ is closed if there is no graph $P'$ such that $P$ is a subgraph of $P'$ and they have the same frequency. In this sense, closed patterns are locally "maximal" patterns that cannot grow any further without losing some instances.

Graph pattern mining [Inokuchi *et al.*, 2002; Yan and Han, 2003; Nijssen and Kok, 2004] follows the Apriori principle: the support of a subgraph is always no less than the support of any of its super-graphs. Thus we can first consider frequent small graphs, and then check whether larger graphs that contain them are still frequent. There are two general approaches to efficient graph pattern mining [Yan and Han, 2003]: Apriori-based approaches extend the Apriori-based candidate generation-and-test approach while pattern-growth approaches grow patterns from

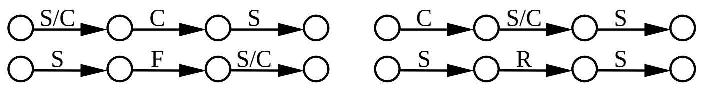


Figure 4.3: Four occurrences of a frequent pattern. S: SaveAs, C: Copy&Paste, F: File Copy, R: File Rename.

a single graph directly. Pattern-growth approaches are usually more efficient, since they avoid some costly operations such as joining two subgraphs into a larger graph. In this chapter, we extend the Gaston algorithm [Nijssen and Kok, 2004]. Recall that we ignore patterns that consist only of email messages and patterns that involve fewer than $k$ resources. The remaining closed patterns correspond to candidate work procedures, except that their arcs are not labeled.

### 4.5.2 Searching For Frequent Activity Paths

The next step is to assign action type labels to the arcs. In principle, we could consider all possible action type assignments and score them to see how many work procedure instances they have. We would then retain only those assignments that had at least the minimum number of instances. However, there are combinatorially many possible assignments. Simply picking the most frequent individual action type for each edge does not necessarily find patterns with many instances. Figure 4.3 shows an example. If we pick the most frequent activity separately for each edge, then the labels $S$, $C$, and $S$ would be the result. However, pattern $(S, C, S)$ only appears once. Instead, the most frequent assignment should be $(C, C, S)$, which appears twice.

**Algorithm 4.1** Search For Frequent Action Label Assignments

```
Require: $P$: frequent pattern, $O$: its occurrences in $D$
 1: $A = \{\langle \emptyset, O \rangle\}$
 2: for each edge $e = u \rightarrow v$ in $P$ do
 3:    $A' = \emptyset$
 4:    for each label $a$ appearing between $u$ and $v$ in $O$ do
 5:       for each set $\langle Acts, Occ \rangle \in A$ do
 6:          Search for occurrence set
                $O' = \{o | Acts \wedge (a \in e) \in o, o \in Occ\}$
 7:          if $|O'| \geq$ the support threshold then
 8:             $A' = A' \cup \{\langle Acts \wedge (e = a), O' \rangle\}$
 9:          end if
10:       end for
11:    end for
12:    if $A' = \emptyset$ then
13:       return fail
14:    else
15:       $A = A'$
16:    end if
17: end for
18: return all activity paths in $A$
```

We employ a dynamic programming approach, as described in the above algorithm, to search for action assignments that appear no less than the minimum support threshold in the information flow graph. The complexity of this algorithm is $O(|P| \cdot m \cdot s)$ where $|P|$ is the number of edges in pattern $P$, $m$ is the maximal number of labels that edges of $P$ have, and $s$ is the largest size of $P$'s frequent action subsets given different numbers of actions. Since $m$ and $s$ are usually small, this algorithm can efficiently find the frequent action label assignments. Note that there is a chance that no assignment of action types to the arcs in $P$ can produce a sufficiently frequent work procedure, in which case we discard $P$.

## 4.6 Workflow Recognition with Mined Models

The mined frequent work procedures are then converted into logical Hidden Markov Models (LHMMs) [Kersting *et al.*, 2006]. We apply those LHMMs for workflow recognition. The observation is fed into a pool of work procedure instance candidates, each corresponding to a logical Hidden Markov Model. If the probability that an LHMM is in the "Begin" state is lower than the threshold, then it becomes "active" and is moved into the pool of active work procedure instances. Meanwhile a new work procedure instance candidate with the same LHMM will be created and added into the candidate pool. If an active work procedure instance reaches the "End" state, it will be removed from the pool of active work procedure instances.

### 4.6.1 Converting Work Procedures into LHMMs

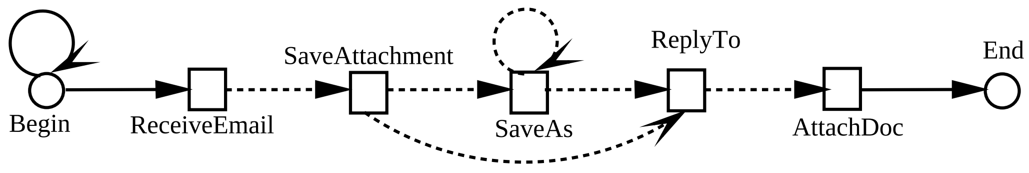
Each mined frequent work procedure is converted into one logical Hidden Markov Model. We create the observation probabilities and transition probabilities according to the following:

- Each edge in the work procedure corresponds to a distinct state in the LHMM. We also create a "Begin" state, an "End" state, and one "Background" states for each edge. The "Background" states correspond to those background actions irrelevant to the work procedures.

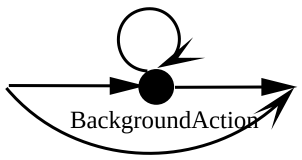
- All observations are treated as deterministic, as in previous LHMM applications [Natarajan *et al.*, 2008]. This means that a "SaveAttachment" state will produce a "SaveAttachment" observation with probability 1, a "SaveAs" state will produce a "SaveAs" observation with probability 1, and so on.

- For transition probabilities, we generate the transitions based on the training set of work procedure instances. Given states $P$ and $Q$, if there is a work procedure instance in which state $Q$ is reached by an information flow edge from state $P$, then we create a transition from state $P$ to state $Q$. The probability $P \rightarrow Q$ is estimated as $(N(P \rightarrow Q) + 1)/(N(P) + \kappa)$, where $N(P)$ is the number of occurrences of state $P$, $N(P \rightarrow Q)$ is the number of observed transitions from $P$ to $Q$ and $\kappa$ is a smoothing factor.

- The probability of staying in the "Begin" state is set to $\alpha$ for all LHMMs, and the probability of statying in a "Background" state is set to $\beta$ for all



(a) Generated LHMM. Each dotted edge can be replaced by the model in Figure 4.4(b).



(b) The model corresponding to the "Background" actions.

Figure 4.4: The logical Hidden Markov Model corresponding to "providing comments on a document".

LHMMs. We set $\alpha = 0.9$ and the default $\beta = 0.8$.

As an example, the work procedure "providing comments on a document" is converted into the logical Hidden Markov Model shown in Figure 4.4(a).

### 4.6.2 Tracking the State of Workflows with LHMMs

We integrate the converted logical Hidden Markov Models into the WARP system [Natarajan *et al.*, 2008] of CALO [SRI, 2009] for workflow recognition. We maintain two pools of LHMMs. The candidate pool stores the LHMMs that have not been "activated". The active pool stores those "active" LHMMs, each corresponding to a work procedure instance that has executed at least one action but has not executed all necessary actions and reached the "End" state.

There are many kinds of events in the TaskTracer event stream. We make an inference only if the type of the event matches at least one edge in a mined work procedure. This is because observations are deterministic, so we can filter out irrelevant events to improve inference efficiency.

Each valid event is fed into every LHMM in both the candidate pool and the active pool. If the probability that an LHMM $L_c$ in the candidate pool is in the state "Begin" is smaller than a specified threshold (the default value is 0.1), then we conclude that the user has started a work procedure instance corresponding to $L_c$. So we move $L_c$ into the active pool. We create another LHMM with the same structure with $L_c$ and add it into the candidate pool to detect future instances of this work procedure.

For each LHMM $L_i$ in the active pool, we search for the non-background state $\hat{s}_i$ which has the maximal posterior probability given the seen observations. We let its corresponding probability be $P_i$. If the largest $P_i$ is larger than a specified threshold (the default value is 0.5), then we conclude that the user is not executing a background action. The LHMM with the largest $P_i$ will be predicted as the one that generated the current observation and the current state is its corresponding non-background state. Otherwise, we predict that the observation is generated by a background action.

If the probability that an LHMM $L_a$ in the active pool is in the state "End" is larger than a specified threshold (the default value is 0.9), then we conclude that the user has completed the corresponding work procedure. We remove this workflow instance from the active pool.

## 4.7 Experimental Results

To evaluate our approach, we conducted experiments based on data from real users. In December, 2007, the TaskTracer system was deployed on Windows machines at SRI International as part of a 3-day data collection exercise. Four staff members participated in an exercise of knowledge work in which they submitted and reviewed papers for a conference, filed travel documents, and prepared quarterly reports. These activities were interleaved with other things (e.g., reading online newspapers). We combined the actions identified through desktop instrumentation with the additional actions described above to produce information flow graphs for each user. We manually analyzed the data and identified all of the work procedures that the users executed. We identified 24 instances of work procedures involving 118 resources and 106 actions.

### 4.7.1 Accuracy of Mining Work Procedures

We applied our learning algorithm to these information flow graphs to discover work procedures. We evaluate the results by measuring the extent to which the discovered work procedures match the manually-identified instances without including false steps or omitting steps.

We measure the accuracy of our algorithm in a leave-one-out cross-validation style. For each person, we hold out his/her data for testing and mine frequent work procedures from the remaining data. Then we examine the test data and predict the subgraphs that exactly match the mined models as the work procedure

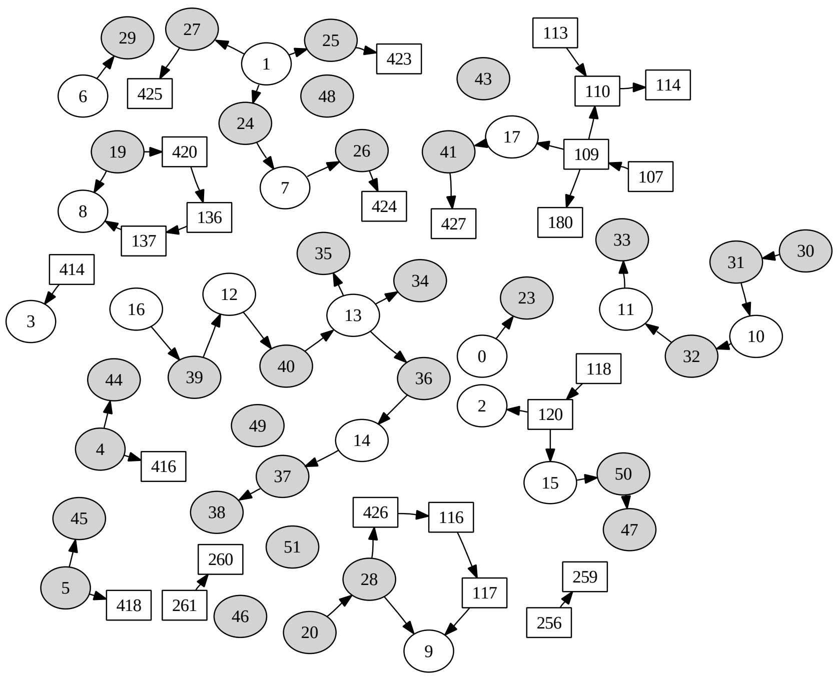
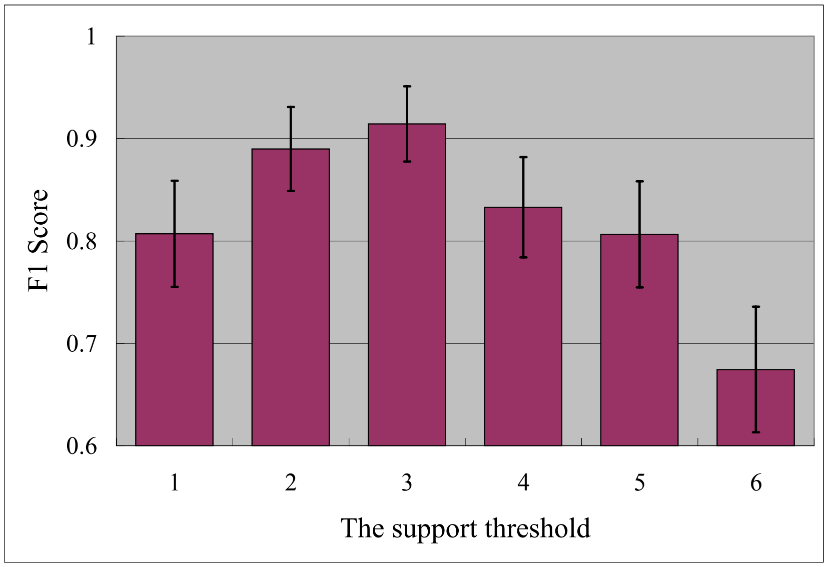


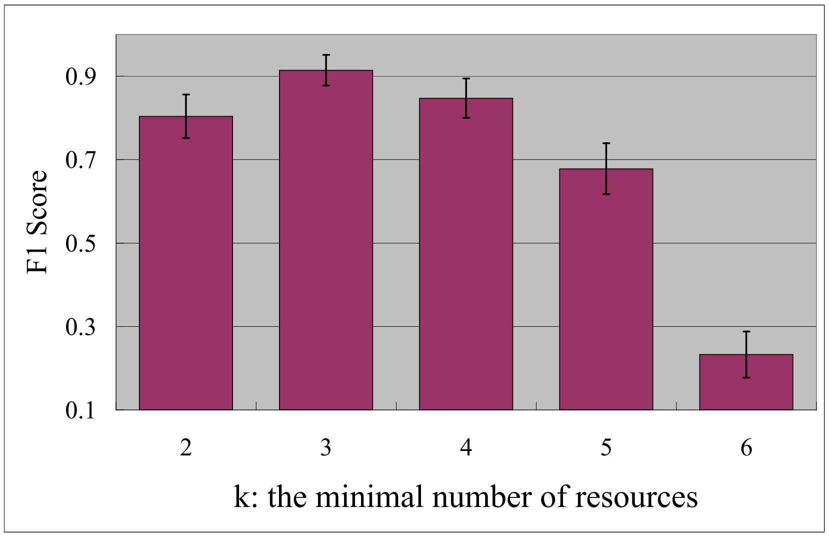
Figure 4.5: Part of a user's resource relationship graph. Shaded circles are received emails. Non-shaded circles are sent emails. Boxes are non-email resources.

instances. We average the *precision* and *recall* values to measure the performance of our algorithm. Precision is defined as the number of true relevant resources and actions in the set of detected occurrences divided by the total number of resources and actions in the set of detected occurrences, and Recall is defined as the number of relevant resources and actions in the set of detected occurrences divided by the total number of true relevant resources and actions (which should have been retrieved by the set of occurrences). We evaluate the algorithms with the F1 score [Joachims, 2001] which is 2×precision×recall /(precision+recall).

A portion of one user's resource relationship graph is shown in Figure 4.5. It contains some interesting work procedures. For example, Email28 from User2 is the



(a) F1 score as a function of the minimum support threshold, with 95% confidence intervals.



(b) F1 score as a function of the minimum number of resources involved in work procedures, with 95% confidence intervals.

Figure 4.6: The accuracy of the mined frequent work procedures.

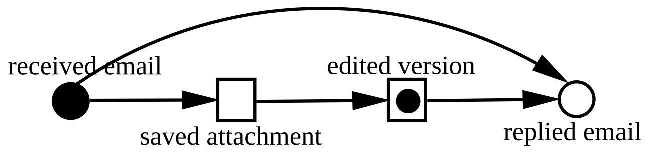


Figure 4.7: The discovered most frequent work procedure: edit and return Document. The box with a dot inside means zero to multiple "SaveAs" resources.

start of a work procedure asking User1 to comment on User2's paper and Email19 from User3 is the start of a work procedure asking User1 to make a contribution to a quarterly report.

Figure 4.6(a) plots F1 as a function of the minimum support threshold of the discovery algorithm. When the threshold is small, the algorithm falsely treats many irrelevant resources and actions as work procedures. This produces high recall but very low precision. As the threshold increases, the number of false work procedures decreases, since such false procedures don't happen very often in the data. The best trade-off between precision and recall is achieved when the threshold is 3. Our approach successfully discovers most work procedures without including many false resources or actions.

Figure 4.6(b) plots F1 as a function of the minimum number $k$ of resources that must be present in the work procedure. When $k$ is small, the algorithm falsely treats some random patterns as work procedures. This produces low precision. Most work procedures involve fewer than 6 resources. Thus the recall is very low when $k$ becomes too large. The best performance is achieved when we require that a work procedure involves at least 3 resources.

The most frequent work procedure we discovered was "edit and return a document", as shown in Figure 4.7. In this procedure, the user receives an email with an attachment, asking him to edit this attachment and return it. So he saves that attachment and begins editing it. He sometimes uses SaveAs to save the edited document with a new name. After editing the file, he replies to the original email message and attaches the edited file. This general work procedure includes at least two cases: "comment on another person's paper" and "contribute to CALO quarterly report". Other mined work procedures are shown in the Appendix.

### 4.7.2 Accuracy of Tracking the States of Workflows

We convert the mined work procedures into Logical HMMs and apply those models to predict the states of the user's workflows. This includes two problems:

- Is an action part of a work procedure instance or just a background action?
- If it belongs to a work procedure instance, which state is it in?

Again, we evaluate the models with precision and recall: precision is defined as the number of work procedure states correctly identified by our algorithm divided by the total number of work procedure states predicted by our algorithm, and recall is defined as the number of work procedure states correctly identified by our algorithm divided by the total number of work procedure states (which should have been predicted).

The evaluation is carried out online: after we predict the workflow state of the current observation, the user provides feedback by correcting any errors. The main purpose of this is to provide a reliable approach to control the number of active workflows.

With the default threshold values, the performance of our algorithm is the following:

- **Recall**: 0.6667
- **Precision**: 0.913
- **F1 score**: 0.7706

We set a high threshold to predict if a new work procedure instance starts. The low recall is mainly caused by the "deterministic" observation rules: most work procedures start by observing an incoming email message. This depends entirely on the "receiving email" action and doesn't analyze the contents of emails at all. As a result, our method often misses the start of a workflow instance, and the user must correct this. There are some other things contributing to the low recall: sometimes there is no information flow connection between the "call for paper" email and the submitted paper. The users were supposed to follow a workflow in which they received the call for papers (with a hyperlink to the template), downloaded the paper template, edited it to create their conference paper, and then submitted it. However, sometimes users just modified some existing paper and then submitted it. We could have considered this as an alternative work procedure to be discovered by the system, but instead we treated this as an error compared to the gold standard work procedure. In the future, we also want to design a better rule to determine if a new work procedure instance has just started. This might be achieved by analyzing the email content.

## 4.8 Discussion

The focus of this chapter is on discovering work procedures frequently executed by users and apply them for workflow recognition. The procedures that we discover do not contain non-deterministic choices or conditionals (except for the resource-action chains). The procedures also do not capture partial order constraints on the steps. One can imagine learning models that include "choice" branches that require the user to choose one of several paths to reach a goal (which would support optional steps and alternative ways of performing a task) and "parallel" branches that require the user to finish a set of parallel paths in order to reach a goal [Medeiros *et al.*, 2003] (which would support partially-ordered actions). With these extensions, different executions of one procedure could lead to different observations or to observations occurring in different orders. With enough data, it should be possible to discover these more complex procedures.

We would also like to integrate our workflow mining and recognition methods into Towel, a smart To-Do item management system [Conley and Carpenter, 2007]. It currently adopts a programming by demonstration approach [Lau *et al.*, 2003; Gervasio *et al.*, 2008] to learn the user's workflow model. It then uses those models to monitor the progress of the user's workflows. We can reduce the user's burden by applying our automatic workflow model mining approach.

# Chapter 5 – Conclusion and Discussion

## 5.1 Summary of Contributions

This dissertation has proposed a set of approaches to recognizing activities. We classify activities into three levels. At the highest level is the kind of weak activity model that TaskTracer [Dragunov *et al.*, 2005] has: an activity is a collection of "resources" (web pages, folders, documents, email messages, email contacts). Typical examples are "Advising student YY", "Project ZZ", and so on. We call such activities *tasks*. The next level is a kind of *workflow* model. A workflow can be thought as a "To-Do" item, consisting of several steps with some temporal constraints. A workflow usually involves an initiation (typically the receipt of an email message with some information such as a URL or an attached file and a request or command), a partially-ordered sequence of steps, and a termination. The "task" above usually consists of several workflows. The third level is the functional *operation*, such as retrieving a paper, saving a file in the "reading" folder, and printing a copy of a file, and so on. A workflow usually consists of several operations. In this thesis, we focus on recognizing the higher levels of activity – "task" and "workflow".

This research is partially motivated by the user requirements of TaskTracer. TaskTracer is an intelligent activity management system [Kaptelinin, 2003] that helps knowledge workers manage their work based on two assumptions: (a) the user's work is organized as a set of ongoing tasks, and (b) each task is associated with a set of resources. TaskTracer collects events generated by the user's computer-visible behavior, including events from MS Office, Internet Explorer, Windows Explorer, and the Windows XP operating system. The user can declare a "current task", and TaskTracer records all resources as they are accessed and associates them with the current declared task. TaskTracer then configures the desktop in various ways to support the user based on the task-resource associations.

We presented two learning systems for predicting the current task of the user. The first system is called TaskPredictor, which predicts the user's current task based on the title and document pathname (or URL) of the window currently in focus. It applies three machine learning techniques in order to "do less" but do it correctly: 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. To reduce false alarms, we proposed to make 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).

The second system described in this thesis is TaskPredictor2, a novel task-switch predictor for the desktop environment. Its user interface displays task switches on a time line that shows accessed resources. This gives the user contextual hints and makes it easy for the user to adjust the time at which a task switch (declared or predicted) actually occurred. Instead of simply reminding the user about the switch, it allows the user to edit the task switch history and thereby adjust resource-task associations. The learning component exploits rich contextual information, including both task-specific features and switch-specific features, to predict task switches. The predictor parameters are trained via a novel online learning algorithm. This algorithm is more accurate, more efficient in memory, and runs 180 times faster than our previous method applied in TaskPredictor.

We also investigated how to estimate the states of the executed workflows. In order to do this, we need a set of workflow models. Traditionally, researchers address this problem by providing easy ways for users to define the workflow procedures themselves (e.g., CoScripter [Leshed *et al.*, 2008]). This thesis investigated how to discover a special kind of workflow model – work procedures from the user desktop activity. A work procedure is defined as a directed labeled graph, with nodes corresponding to the resources involved and edges corresponding to the actions executed. To discover work procedures frequently executed, we first built an information flow graph of resources based on the provenance information captured by TaskTracer (and manually extended by analyzing resource content). We then applied a two-phase algorithm to find frequently-occurring work procedures in these graphs. The mined models were converted into logical Hidden Markov models [Kersting *et al.*, 2006] for workflow recognition. Experimental results show that our approach can accurately discover almost all of the known work procedure instances without introducing many false work procedures.

## 5.2 Future Work

I have tried hard to make machine learning systems practical in the real world. In the future, there are many interesting research problems that I like to explore.

### 5.2.1 Unsupervised Version of TaskTracer

In the current version of TaskTracer, we rely on the user to define a set of tasks. Then the user declares the current task and TaskTracer monitors the events happening in the desktop. By building associations between resources and tasks, we aggregate resources into a set of groups, i.e., "tasks". In the future, I would like to explore how to aggregate resources into clusters without explicitly defining any task.

Researchers have tried to automatically cluster resources into activities (e.g., [Mitchell *et al.*, 2006]). Such approaches are usually carried out by analyzing the resource content and assigning resources with the similar content to the same cluster. By monitoring the events happening in the desktop, we can do a much better job than content analysis. For example, if the user copies and pastes some content from resource $A$ to resource $B$, then it is a strong indicator that $A$ and $B$ might belong to the same cluster. If resource $A$ is accessed just after $B$ is accessed, it is also a strong indicator that they might belong to the same cluster.

### 5.2.2 Natural Communications between Users and Computers

I am also interested in investigating new human-computer-interaction solutions for knowledge workers. Desktop users have to handle thousands of files (documents, pictures, video clips, and so on.). It is desirable that an intelligent personal assistant helps the user find the information in a shorter time and in an easier way than current desktop search tools.

To find the resource they want, users typically conduct a search using desktop tools. However, such search tools cannot search based on the meaning of the resources. For example, "I want to find some pictures in which my friends and I appear and I am smiling". As a very first step, we can support an easy way for the user to annotate the resources and then analyze the semantic meaning of those annotations. Then we can analyze the user's search query and map to the corresponding resources. Such annotations also provide some "unstructured" labels for the resources. It would be interesting to abstract keywords from the annotations and train some classifiers to automatically assign annotations to new resources.

### 5.2.3 Personalizing the Notification

When to interrupt the user and issue the switch alert is important: the user could be busy doing something when the system is confident about a task switch. The interruption cost can be very high. Thus, we should defer the alert to an appropriate time when the user is interruptible. Different users have different working habits, and their tolerance to interruptions can be quite different.

Currently, the system uses simple rules to control when to issue the switch alert. It does not fit different users' requirements. We can make the notification controller adaptive by applying reinforcement learning technology. The notification control problem can be readily formalized as a Partially Observable Markov Decision Process (POMDP) [Monahan, 1982; Cassandra *et al.*, 1994].

To succeed in practice, the system must employ a strong initial policy, and must efficiently exploit the sparse data. We can initialize the policy with other users' data. We can rely on a parametric form of the value function. We can compute the gradient with respect to the parameters of the average reward, and then improve the policy by adjusting the parameters in the gradient direction. Such policy gradient search methods have been shown to be very effective in finding good policies [Baxter and Bartlett, 2001; Ng *et al.*, 2004].

There have been some works attempting to employ Markov Decision Process models to design intelligent assistants (e.g., [Singh *et al.*, 2002; Rudary *et al.*, 2004; Shani *et al.*, 2005; Boger *et al.*, 2005; Fern *et al.*, 2007a]). When reinforcement learning is applied, they typically adopt Q-Learning because of its simplicity. For adaptive user interfaces, we will need a more efficient learning algorithm that can learn a reasonable policy within a limited number of trials.

### 5.2.4 Avoiding Unnecessary Inference Computations

In most machine learning settings, the system first computes all necessary feature values and then makes inferences based on them. However, it is not really necessary to compute all feature values: usually the decision can be made based on a small subset of the feature values. For an interactive intelligent desktop application, we need a very efficient inference method that can avoid unnecessary computations as much as possible.

If we use decision trees as the learning algorithm, the problem is simple: we first compute the value of the root splitting condition, and then go to the corresponding child. We continue compute the value of the node splitting conditions until we reach a leaf. When we adopt the kernel methods such as SVMs, the problem becomes more complicated. The SVM learns a decision boundary as the decision tree does, however it is more difficult to determine which features are relevant when making decisions (with kernelized SVMs). One approach is to convert this decision boundary into a set of nested decision rules, so the expected count of feature value computations is minimal. Another approach is to treat the inference process as a functional expression, and use the optimization technique of functional programming languages to avoid unnecessary computations [Asperti and Guerrini, 1998].

### 5.2.5 Using Active Sequential Learning to Reduce Noise

To collect the labeled training data, we rely on the user to indicate the task by selecting the task 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, previously we employed an active EM algorithm that combines the EM algorithm and active learning [Shen and Dietterich, 2007]. This can significantly improve the prediction precision. However, this algorithm is based on the i.i.d assumption. We could improve its performance even further if we could exploit sequential information.

There has been some work on active sequential learning [Anderson and Moore, 2005; Culotta and McCallum, 2005]. These approaches ask the user to label a specific instance or an entire sequence. We can also ask the user to indicate whether two instances belong to the same segment. Such queries usually require much less effort from the user: the user just needs to answer yes or no, instead of contemplating the real labels of instances.

### 5.2.6 Transferring Other Users' Data

Knowledge workers usually have similar working patterns, especially if they are in the same group and working on the same project. Currently, our intelligent system only utilizes an individual user's personal data and just ignores other users' data. We could improve the system's performance by exploiting other users' data.

As a use case, we can think about the notification controller: the system should decide an appropriate time when the user is interruptible to issue the switch alert. If we adopt the reinforcement learning approach, the system must learn an appropriate policy quickly enough to succeed in practice. The simplest solution is to use the policy learned from other users' data as the initial policy, and then update it as usual. We can also try a hierarchical Bayesian approach: we can assume the optimal policy for each user is generated from a "parent policy", and update our belief about a specific user's optimal policy according to the data seen so far.

# APPENDICES

**Proof of Lemma 1**

**Proof** The Lagrangian of the optimization problem in (3.2) is

$$L(\Lambda, \xi, \tau) = \frac{1}{2}\left\|\Lambda - \Lambda_t\right\|_2^2 + C\xi^2 + \frac{\alpha}{2}\left\|\Lambda\right\|_2^2 \\ + \tau(1 - \Lambda \cdot \mathbf{Z}_t - \xi), \tag{1}$$

where $\tau \geq 0$ is the Lagrange multiplier. Differentiating this Lagrangian with respect to the elements of $\Lambda$ and setting the partial derivative to zero gives

$$\Lambda = \frac{1}{1+\alpha}\Lambda_t + \frac{\tau}{1+\alpha}\mathbf{Z}_t. \tag{2}$$

Differentiating the Lagrangian with respect to $\xi$ and setting the partial derivative to zero gives

$$\xi = \frac{\tau}{2C}. \tag{3}$$

Expressing $\xi$ as above and replacing $\Lambda$ in Eq 1 with Eq 2, the Lagrangian becomes

$$\begin{aligned} L(\tau) =& \frac{1}{2}\left\|\frac{\tau}{1+\alpha}\mathbf{Z}_t - \frac{\alpha}{1+\alpha}\Lambda_t\right\|_2^2 + \frac{\tau^2}{4C} \\ &+ \frac{\alpha}{2}\left\|\frac{\tau}{1+\alpha}\mathbf{Z}_t + \frac{1}{1+\alpha}\Lambda_t\right\|_2^2 \\ &+ \tau\left(1 - \frac{\Lambda_t \cdot \mathbf{Z}_t}{1+\alpha} - \frac{\tau\left\|\mathbf{Z}_t\right\|_2^2}{1+\alpha} - \frac{\tau}{2C}\right). \end{aligned}$$

By setting the derivative of the above to zero, we get

$$1 - \frac{\tau}{2C} - \frac{\tau}{1+\alpha}\|\mathbf{Z}_t\|_2^2 - \frac{(\Lambda_t \cdot \mathbf{Z}_t)}{1+\alpha} = 0 \tag{4}$$

$$\Rightarrow \quad \tau = \frac{1 - (\Lambda_t \cdot \mathbf{Z}_t) + \alpha}{\|\mathbf{Z}_t\|_2^2 + \frac{1+\alpha}{2C}}. \tag{5}$$

Combining Eq 2 and Eq 5, completes the proof. ∎

**Proof of Theorem 1**

**Proof** Let $\Delta_t = \|\Lambda_t - \mathbf{u}\|_2^2 - \|\Lambda_{t+1} - \mathbf{u}\|_2^2$. We can prove the bound by lower and upper bounding $\sum_t \Delta_t$.

Since $\Lambda_0$ is a zero vector and the norm is always non-negative, $\sum_t \Delta_t = \|\Lambda_0 - \mathbf{u}\|_2^2 - \|\Lambda_T - \mathbf{u}\|_2^2 \leq \|\Lambda_0 - \mathbf{u}\|_2^2 = \|\mathbf{u}\|_2^2$.

Obviously, only if $t \in I_T$, $\Delta_t \neq 0$. We will only consider this case here. Let $\Lambda'_t = \Lambda_t + \tau_t \mathbf{Z}_t$, $\Lambda_{t+1} = \frac{1}{1+\alpha}\Lambda'_t$. $\Delta_t$ can be rewritten as

$$\begin{aligned} &(\|\Lambda_t - \mathbf{u}\|_2^2 - \|\Lambda'_t - \mathbf{u}\|_2^2) + (\|\Lambda'_t - \mathbf{u}\|_2^2 - \|\Lambda_{t+1} - \mathbf{u}\|_2^2) \\ &= \delta_t + \epsilon_t. \end{aligned} \tag{6}$$

We will lower bound both $\delta_t$ and $\epsilon_t$. Let $\psi^2 = (1+\alpha)/2C$. For $\delta_t$, we have

$$\begin{aligned}
\delta_t =& \|\Lambda_t - \mathbf{u}\|_2^2 - \|\Lambda_t + \tau_t \mathbf{Z}_t - \mathbf{u}\|_2^2 && (7)\\
=& \|\Lambda_t - \mathbf{u}\|_2^2 - \|\Lambda_t - \mathbf{u}\|_2^2 \\
& - 2\tau_t \mathbf{Z}_t \cdot (\Lambda_t - \mathbf{u}) - \|\tau_t \mathbf{Z}_t\|_2^2 && (8)\\
\geq& 2\tau_t \ell_t - 2\tau_t \ell_t^* - \tau_t^2 \|\mathbf{Z}_t\|_2^2 && (9)\\
\geq& 2\tau_t \ell_t - 2\tau_t \ell_t^* - \tau_t^2 \|\mathbf{Z}_t\|_2^2 - (\psi\tau_t - \ell_t^*/\psi)^2 && (10)\\
=& 2\tau_t \ell_t - \tau_t^2 (\|\mathbf{Z}_t\|_2^2 + \psi^2) - (\ell_t^*)^2/\psi^2. && (11)
\end{aligned}$$

We get Eq 10 because $(\psi\tau_t - \ell_t^*/\psi)^2 \geq 0$. Plugging the definition of $\tau_t$ and considering $\ell_t \geq 1$, we get

$$\begin{aligned}
\delta_t \geq& \frac{\ell_t^2 - \alpha^2}{\|\mathbf{Z}_t\|_2^2 + \frac{1+\alpha}{2C}} - \frac{2C}{1+\alpha}(\ell_t^*)^2 && (12)\\
\geq& \frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{2C}{1+\alpha}(\ell_t^*)^2. && (13)
\end{aligned}$$

For $\epsilon_t$, we have

$$\epsilon_t = (1 - \frac{1}{(1+\alpha)^2}) \|\Lambda_t'\|_2^2 - 2(1 - \frac{1}{1+\alpha})\Lambda_t' \cdot \mathbf{u}. \qquad (14)$$

Using the fact that $\|\mathbf{u} - \mathbf{v}\|_2^2 \geq 0$ which equals to $\|\mathbf{u}\|_2^2 - 2\mathbf{u}\cdot\mathbf{v} \geq -\|\mathbf{v}\|_2^2$, we get

$$(1 - \frac{1}{(1+\alpha)^2}) \left\| \Lambda'_t \right\|_2^2 - 2(1 - \frac{1}{1+\alpha})\Lambda'_t \cdot \mathbf{u} \tag{15}$$

$$\geq - \frac{1 - \frac{1}{1+\alpha}}{1 + \frac{1}{1+\alpha}} \left\| \mathbf{u} \right\|_2^2 = -\frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2 . \tag{16}$$

Using Eq 13 and 16, we get

$$\sum_{t=0}^{T} \Delta_t = \sum_{t \in I_T} \Delta_t \tag{17}$$

$$\geq \sum_{t \in I_T} \left( (\frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{2C}{1+\alpha}(\ell_t^*)^2) - \frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2 \right) \tag{18}$$

$$= m(\frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2) - \sum_{t \in I_t} \frac{2C}{1+\alpha}(\ell_t^*)^2 \tag{19}$$

$$\geq m(\frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2) - \frac{2C}{1+\alpha} \sum_{t=0}^{T} (\ell_t^*)^2. \tag{20}$$

Since $\sum_t \Delta_t \leq \left\| \mathbf{u} \right\|_2^2$, we have

$$m(\frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2) - \frac{2C}{1+\alpha} \sum_{t=0}^{T} (\ell_t^*)^2 \leq \left\| \mathbf{u} \right\|_2^2 . \tag{21}$$

Since $h = \frac{1-\alpha^2}{R^2 + \frac{1+\alpha}{2C}} - \frac{\alpha}{2+\alpha} \left\| \mathbf{u} \right\|_2^2 > 0$, we get the result in the theorem. ∎

**Mined Work Procedures**

There are five work procedures that can be reliably discovered in the leave-one-out cross-validation style:

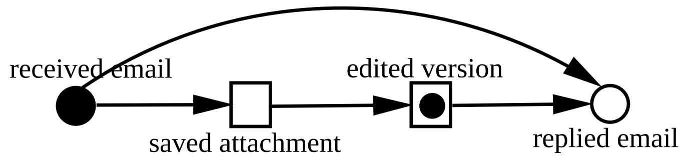


Figure 1: The discovered frequent work procedure: edit and return Document. The box with a dot inside means zero to multiple "SaveAs" resources. User Person1 receives an email with an attachment, asking him/her to edit this attachment and return it. So Person1 saves that attachment and begins editing it. Person1 sometimes uses SaveAs to save the edited document with a new name. After editing the file, Person1 replies to the original email message and attaches the edited file.

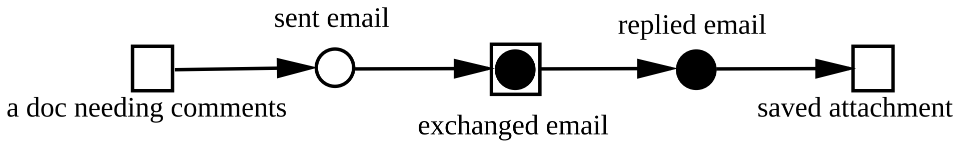


Figure 2: The discovered frequent work procedure: request comments from another user. The box with a dot inside means zero to multiple exchanged email messages. User Person1 attaches a document to an email message and sends it to user Person2. Person1 might exchange several email messages with Person2 regarding the document. After the final feedback comes in, Person1 saves it into the right folder.

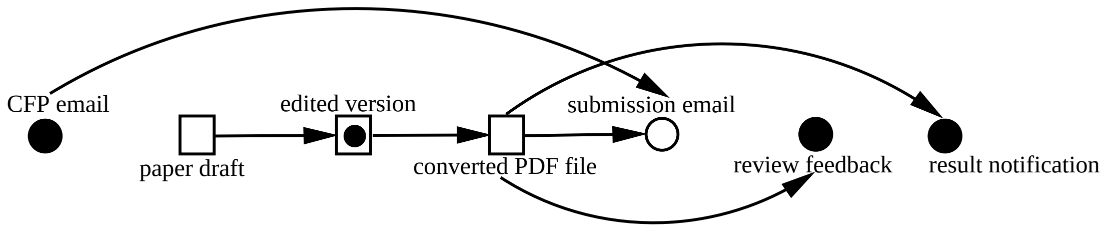


Figure 3: The discovered frequent work procedure: submit a paper to a conference. User Person1 receives a CFP email. So Person1 prepares a doc file and then converts it into a pdf file. Person1 then submits the pdf file through email. He/She will receive an email message asking for the feedback regarding the review and an email message regarding the final decision.

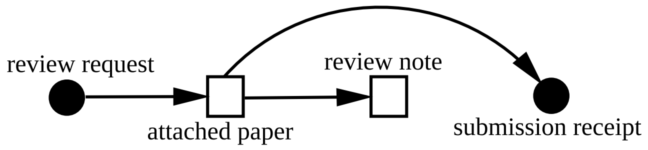


Figure 4: The discovered frequent work procedure: provide reviews on a paper. User Person1 receives an email asking him/her to provide reviews on the attachment. Person1 saves this pdf file and creats a doc file to write down the reviews. A notification email will be sent out once Person1 submits the reviews.

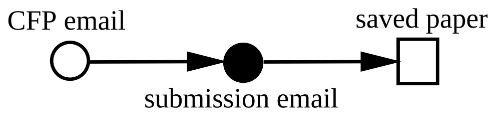


Figure 5: The discovered frequent work procedure: call for papers. User Person1 sends out an email message, calling for papers. When the submission comes in, Person1 saves the attached pdf file into the right folder.

# Bibliography

[Agrawal *et al.*, 1998] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Proc. of Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

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

[Amershi and Conati, 2007] Saleema Amershi and Cristina Conati. Unsupervised and supervised machine learning in user modeling for intelligent learning environments. In *Proc. of IUI-07*, pages 72–81, 2007.

[Anderson and Moore, 2005] B. Anderson and A. Moore. Active learning for hidden markov models: objective functions and algorithms. In *Proc. of ICML-05*, pages 9–16, 2005.

[Andrews *et al.*, 2004] 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.

[Asperti and Guerrini, 1998] A. Asperti and S. Guerrini. *The optimal implementation of functional programming languages.* Cambridge University Press, New York, NY, USA, 1998.

[Bao *et al.*, 2006] X. Bao, J. Herlocker, and T. G. Dietterich. Fewer clicks and less frustration: Reducing the cost of reaching the right folder. In *Proc. of IUI-06*, pages 178–185, 2006.

[Baxter and Bartlett, 2001] J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

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

[Bellotti *et al.*, 2003] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI-03*, pages 345 – 352, 2003.

[Blanc-Brude and Scapin, 2007] Tristan Blanc-Brude and Dominique L. Scapin. What do people recall about their documents?: implications for desktop search tools. In *Proc. of IUI-07*, pages 102–111, 2007.

[Boger *et al.*, 2005] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proc. IJCAI-05*, 2005.

[Campbell and Bennett, 2001] Colin Campbell and Kristin P. Bennett. A linear programming approach to novelty detection. In *NIPS 13*. 2001.

[Cassandra *et al.*, 1994] A. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI-94*, volume 2, pages 1023–1028, 1994.

[Chang and Lin, 2001] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[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.

[Chiu *et al.*, 2003] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In *Proc. of KDD-03*, pages 493–498, 2003.

[Cohen, 2004] William W. Cohen. *Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data*, 2004. Software available at http://minorthird.sourceforge.net.

[Conley and Carpenter, 2007] Kenneth Conley and James Carpenter. Towel: Towards an intelligent to-do list. In *The 2007 AAAI Spring Symposium on Interaction Challenges for Intelligent Assistants*, pages 26–32, 2007.

[Crammer *et al.*, 2006] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[Culotta and McCallum, 2005] A. Culotta and A. McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI-05*, pages 746–751, 2005.

[Czerwinski *et al.*, 2004] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. A diary study of task switching and interruptions. In *Proc. of CHI'04*, pages 175–182, 2004.

[Dietterich and Bui, 2006] T.G. Dietterich and H. Bui. Use cases for the activity activity. Technical Report CALO, Oregon State University & SRI, September 2006.

[Dragunov *et al.*, 2005] 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*, pages 75–82, 2005.

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

[El-Ramly *et al.*, 2002] Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson. From run-time behavior to usage scenarios: an interaction-pattern mining approach. In *Proc. of KDD-02*, pages 315–324, 2002.

[Ellis *et al.*, 2006] Clarence A. Ellis, Kwang-Hoon Kim, and Aubrey J. Rembert. Workflow mining: Definitions, techniques, and future directions. *Workflow Handbook*, pages 213–228, 2006.

[Fawcett and Provost, 1999] T. Fawcett and F. Provost. Activity monitoring: notice interesting changes in behavior. In *Proc. of KDD-99*, pages 53–62, 1999.

[Felzenszwalb *et al.*, 2004] Pedro F. Felzenszwalb, Daniel P. Huttenlocher, and Jon M. Kleinberg. Fast algorithms for large-state-space HMMs with applications to web usage analysis. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Fern *et al.*, 2007a] A. Fern, S. Natarajan, K. Judah, and P. Tadepalli. A decision-theoretic model of assistance. In *Proc. IJCAI-07*, pages 1879–1884, 2007.

[Fern *et al.*, 2007b] Xiaoli Z. Fern, Chaitanya Komireddy, and Margaret Burnett. Mining interpretable human strategies: A case study. In *Proc. of ICDM-07*, pages 475–480, 2007.

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

[Frew *et al.*, 2008] James Frew, Dominic Metzger, and Peter Slaughter. Automatic capture and reconstruction of computational provenance. *Concurr. Comput. : Pract. Exper.*, 20(5):485–496, 2008.

[Gajos and Weld, 2005] Krzysztof Gajos and Daniel S. Weld. Preference elicitation for interface optimization. In *Proc. of UIST-05*, pages 173–182, 2005.

[Gajos *et al.*, 2007] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proc. of UIST-07*, pages 231–240, 2007.

[Gervasio *et al.*, 2008] Melinda Gervasio, Thomas J. Lee, and Steven Eker. Learning email procedures for the desktop. In *Proc. of AAAI 2008 Workshop on Enhanced Messaging*, 2008.

[Geyer *et al.*, 2003] W. Geyer, J. Vogel, L. Cheng, and M. Muller. Supporting activity-centric collaboration through peer-to-peer shared objects. In *SIGGROUP-03*, pages 115–124, 2003.

[Geyer *et al.*, 2006] W. Geyer, M. Muller, M. Moore, E. Wilcox, L.-T. Cheng, B. Brownholtz, C. Hill, , and D. R. Millen. Activityexplorer: Activity-centric collaboration from research to product. *IBM Systems Journal - Special Issue on Business Collaboration*, 45(4):713–738, 2006.

[Gluck *et al.*, 2007] Jennifer Gluck, Andrea Bunt, and Joanna McGrenere. Matching attentional draw with utility in interruption. In *Proc. of CHI-07*, pages 41–50, 2007.

[Gonçalves and Jorge, 2004] Daniel Gonçalves and Joaquim A. Jorge. Describing documents: what can users tell us? In *Proc. of IUI-04*, pages 247–249, 2004.

[Gonzalez and Mark, 2004] V. M. Gonzalez and G. Mark. "constant, constant, multi-tasking craziness": managing multiple working spheres. In *CHI-04*, pages 113–120, 2004.

[Greco *et al.*, 2005] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, and Domenico Sacca. Mining and reasoning on workflows. *IEEE Trans. on Knowl. and Data Eng.*, 17(4):519–534, 2005.

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

[Haigh and Yanco, 2002] K. Haigh and H. A. Yanco. Automation as caregiver: a survey of issues and technologies. In *Proc. of the AAAI-02 workshop "Automation as Caregiver"*, pages 39–53, 2002.

[Hamid *et al.*, 2005] Raffay Hamid, Siddhartha Maddi, Amos Johnson, Aaron Bobick, Irfan Essa, and Charles Isbell. Unsupervised activity discovery and characterization from event-streams. In *Proc. of UAI-05*, 2005.

[Horvitz *et al.*, 1998] 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.

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

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

[Horvitz, 1999] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proc. of CHI-99*, pages 159–166, 1999.

[Inokuchi *et al.*, 2002] Akihiro Inokuchi, Takashi Washio, Kunio Nishimura, and Hiroshi Motoda. A fast algorithm for mining frequent connected graphs. Technical Report RT0448, IBM Research, 2002.

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

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

[Kersting *et al.*, 2006] K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.

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

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

[Lam *et al.*, 2002] Derek Lam, Steven L. Rohall, Chris Schmandt, and Mia K. Stern. Exploiting e-mail structure to improve summarization. In *Proc. of CSCW-02*, 2002.

[Lau *et al.*, 2003] Tessa Lau, Steven A. Wolfman, Pedro Domingos, and Daniel S. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.

[Lau *et al.*, 2004] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In *Proc. of IUI-04*, pages 109–116, 2004.

[Leshed *et al.*, 2008] Gilly Leshed, Eben Haber, Tara Matthews, and Tessa Lau. Coscripter: Automating and sharing how-to knowledge in the enterprise. In *CHI 2008*, 2008.

[Lin *et al.*, 2002] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. In *the 2nd workshop on temporal data mining at the 8th SIGKDD*, pages 53–68, 2002.

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

[Mark *et al.*, 2005] G. Mark, V. M. Gonzalez, and J. Harris. No task left behind? examining the nature of fragmented work. In *Proc. of CHI-05*, pages 321 – 330, 2005.

[McCallum *et al.*, 2000] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. of ICML-00*, pages 591–598, 2000.

[Medeiros *et al.*, 2003] A. K. A. De Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Workflow mining: Current status and future directions. In *CoopIS/DOA/ODBASE, LNCS-2888*, pages 389–406, 2003.

[Minnen *et al.*, 2007a] David Minnen, Charles Isbell, Irfan Essa, and Thad Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *Proc. of AAAI-07*, pages 615–620, 2007.

[Minnen *et al.*, 2007b] David Minnen, Thad Starner, Irfan Essa, and Charles Isbell. Improving activity discovery with automatic neighborhood estimation. In *Proc. of IJCAI-07*, pages 6–12, 2007.

[Mitchell *et al.*, 2006] 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.

[Monahan, 1982] G. E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Moore *et al.*, ] M. Moore, M. Estrada, T. Finley, M. Muller, and W. Geyer. Next generation activity-centric computing. In *Demo at ACM CSCW 2006.*

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

[Muniswamy-Reddy *et al.*, 2006] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proc. of the annual conference on USENIX-06*, 2006.

[Natarajan *et al.*, 2008] Sriraam Natarajan, Hung H. Bui, Prasad Tadepalli, Kristian Kersting, and Weng-Keen Wong. Logical hierarchical hidden Markov models for modeling user activities. In *Proc. of ILP-08*, pages 192–209, 2008.

[Ng *et al.*, 2004] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in NIPS 16*. 2004.

[Nijssen and Kok, 2004] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proc. of KDD-04*, pages 647–652, 2004.

[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.

[Philipose *et al.*, 2003] 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.

[Platt, 1999] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 1999.

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

[Quan and Karger, 2003] D. Quan and D. Karger. Haystack: Metadata-enabled information management. In *UIST-2003.*, 2003.

[Rattenbury and Canny, 2007] Tye Rattenbury and John Canny. Caad: an automatic task support system. In *Proc. of CHI-07*, pages 687–696, 2007.

[Rosenblatt, 1988] F. Rosenblatt. The Perceptron: a probabilistic model for information storage and organization in the brain. *Neurocomputing: foundations of research*, pages 89–114, 1988.

[Rudary *et al.*, 2004] M. Rudary, S. Singh, and M. E. Pollack. Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning. In *Proc. of ICML-04*, pages 91–98, 2004.

[Schölkopf *et al.*, 2000] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *NIPS 11*. 2000.

[Shah *et al.*, 2007] Sam Shah, Craig A. N. Soules, Gregory R. Ganger, and Brian D. Noble. Using provenance to aid in personal file search. In *Proc. of the USENIX-07*, 2007.

[Shani *et al.*, 2005] G. Shani, D. Heckerman, and R. Brafman. An MDP-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

[Shen and Dietterich, 2007] J. Shen and T. G. Dietterich. Active EM to reduce noise in activity recognition. In *Proc. of IUI-07*, pages 132–140, 2007.

[Shen *et al.*, 2006] J. Shen, L. Li, T. G. 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.

[Shen *et al.*, 2007] J. Shen, L. Li, and T. G. Dietterich. Real-time detection of task switches of desktop users. In *Proc. of IJCAI-07*, pages 2868–2873, 2007.

[Shen *et al.*, 2008] Jianqiang Shen, Werner Geyer, Michael Muller, Casey Dugan, Beth Brownholtz, and David R Millen. Automatically finding and recommending resources to support knowledge workers' activities. In *Proc. of IUI-08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 207–216, 2008.

[Shen *et al.*, 2009a] Jianqiang Shen, Erin Fitzhenry, , and T. Dietterich. Discovering frequent work procedures from resource connections. In *Proc. of IUI-09: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 277–285, 2009.

[Shen *et al.*, 2009b] Jianqiang Shen, J. Irvine, X. Bao, M. Goodman, S. Kolibaba, A. Tran, F. Carl, B. Kirschner, S. Stumpf, and T. Dietterich. Detecting and correcting user activity switches: Algorithms and interfaces. In *Proc. of IUI-09: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 117–125, 2009.

[Singh *et al.*, 2002] S. Singh, D. Litman, and M. Kearns. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research*, 2002.

[Smith *et al.*, 2003] Greg Smith, Patrick Baudisch, George Robertson, Mary Czerwinski, Brian Meyers, Daniel Robbins, and Donna Andrews. Groupbar: The taskbar evolved. In *Proc. of OZCHI 2003*, pages 34–43, 2003.

[SRI, 2009] SRI. *CALO: Cognitive Assistant that Learns and Organizes*, 2009. Information available at http://caloproject.sri.com/.

[Tanaka *et al.*, 2005] Yoshiki Tanaka, Kazuhisa Iwamoto, and Kuniaki Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2-3):269–300, 2005.

[Weld *et al.*, 2003] Daniel S. Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically personalizing user interfaces. In *Proc. of IJCAI-03*, pages 1613–1619, 2003.

[Wu *et al.*, 2004] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. In *NIPS 16*. 2004.

[Yan and Han, 2003] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proc. of KDD-03*, pages 286–295, 2003.

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