Machine Learning Methods for High Level Cyber Situation Awareness

Thomas G. Dietterich, Xinlong Bao, Victoria Keiser and Jianqiang Shen

1 Introduction

Cyber situation awareness needs to operate at many levels of abstraction. In this chapter, we discuss situation awareness at a very high level—the behavior of desktop computer users. Our goal is to develop an awareness of what desktop users are doing as they work. Such awareness has many potential applications including

- providing assistance to the users themselves,
- providing additional contextual knowledge to lower-level awareness components such as intrusion detection systems, and
- detecting insider attacks.

The work described here grew out of the TaskTracer system, which is a Microsoft Windows add-in that extends the Windows desktop user interface to become a "project-oriented" user interface. The basic hypothesis is that the user's time at the desktop can be viewed as multi-tasking among a set of active projects. TaskTracer attempts to associate project "tags" with each file, folder, web page, email message, and email address that the user accesses. It then exploits these tags to provide project-oriented assistance to the user.

Thomas G. Dietterich
Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR, 97331 USA, e-mail: tgd@cs.orst.edu

Xinlong Bao
Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR, 97331 USA, e-mail: bao@eecs.oregonstate.edu

Victoria Keiser
Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR, 97331 USA, e-mail: baileyvi@eecs.oregonstate.edu

Jianqiang Shen
Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR, 97331 USA, e-mail: shenj@eecs.oregonstate.edu

To do this, TaskTracer inserts instrumentation into many Windows programs including Microsoft Outlook, Word, Excel, Powerpoint, Internet Explorer, and Windows Explorer (the file browser). This instrumentation captures events at a semantically-meaningful level (e.g., open excel file, navigate to web page, reply to email message) rather than at the level of system calls or keystrokes. This instrumentation also allows TaskTracer to capture so-called “provenance events” that record information flow between one object and another, such as copy-paste, attach file to email message, download file from web page, copy file from flash drive, and so on. These provenance events allow us to automatically discover and track user workflows.

This chapter begins with a description of the TaskTracer system, the instrumented events that it collects, and the benefits it provides to the user. We then discuss two forms of situation awareness that this enables. The first is to track the current project of the user. TaskTracer applies a combination of user interface elements and machine learning methods to do this. The second is to discover and track the workflows of the user. We apply graph mining methods to discover workflows and a form of hidden Markov model to track those workflows. The chapter concludes with a discussion of future directions for high level cyber situation awareness.

2 The TaskTracer System

The goal of the TaskTracer system is to support multi-tasking and interruption recovery for desktop users. Several studies (e.g., [7]) have documented that desktop knowledge workers engage in continual multi-tasking. In one study that we performed [3], we found that the median time to an interruption is around 20 minutes and the median time to return back to the interrupted project is around 15 minutes. Often, many hours or days can pass between periods when the user works on a particular project. These longer interruptions require even more assistance to find the relevant documents, web pages, and email messages.

To provide assistance to knowledge workers, the TaskTracer attempts to associate all files, folders, email messages, email contacts, and web pages with user-declared projects. We will refer to these various data objects as “resources”. To use TaskTracer, the user begins by defining an initial hierarchy of projects. To be most effective, a project should be an ongoing activity such as teaching a class (“CS534”), working on a grant proposal (“CALO”), or performing an ongoing job responsibility (“Annual performance reviews”). Projects at this level of abstraction will last long enough to provide payoff to the user for the work of defining the project and helping with the resource-to-project associations.

2.1 Tracking the User’s Current Project

Once the projects are defined, TaskTracer attempts to infer the user’s current project as the user works. Three methods are employed to do this. First, the user can directly declare to the system his or her current project. Two user interface components support this. One is a drop-down combo-box in the Windows TaskBar that allows the user to type a project name (with auto-completion) or select a project (with arrow keys or the mouse) to declare as the current project (see Figure 1). Another user interface component is a pop-up menu (accessed using the keystroke Control + Backquote) of the 14 most recently used projects (see Figure 2). This supports rapid switching between the set of current projects.

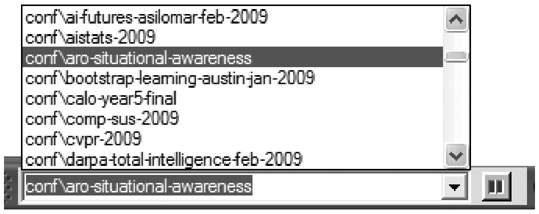

conf\ai-futures-asilomar-feb-2009
conf\aistats-2009
conf\aro-situational-awareness
conf\bootstrap-learning-austin-jan-2009
conf\calo-year5-final
conf\comp-sus-2009
conf\cvpr-2009
conf\darpa-total-intelligence-feb-2009
conf\aro-situational-awareness

Fig. 1 The Windows TaskBar component for declaring the user’s current project.

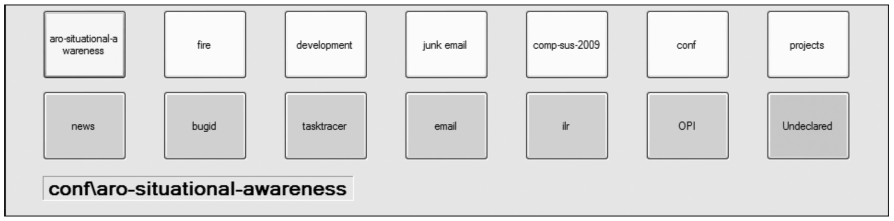


Fig. 2 The pop-up menu that shows the 14 most recently-used projects. The mouse and arrow keys can be used to select the project to switch to.

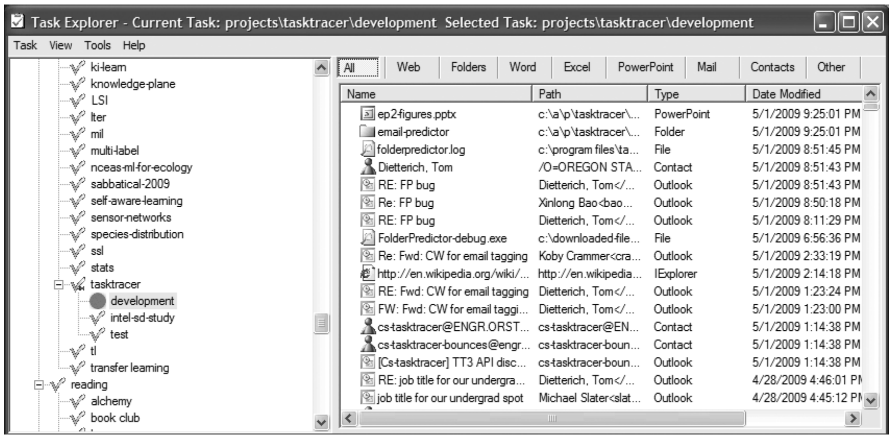
The second way that TaskTracer tracks the current project of the user is to apply machine learning methods to detect project switches based on desktop events. This will be discussed below.

The third method for tracking the user’s current project is based on applying machine learning methods to tag incoming email messages by project. When the user opens an email message, TaskTracer automatically switches the current project to be the project of the email message. Furthermore, if the user opens or saves an email attachment, the associated file is also associated with that project. The user can, of course, correct tags if they are incorrect, and this provides feedback to the email tagger. We will discuss the email tagger in more detail in the next section.

TaskTracer exploits its awareness of the current project to associate each new resource with the user's current project. For example, if the user creates a new Word file or visits a new web page, that file or web page is associated with the current project. This concept of automatically associating resources with a current project (or activity) was pioneered in the UMEA system [9].

2.2 Assisting the User

How does all of this project association and tagging support multi-tasking and interruption recovery? TaskTracer provides several user interface components to do this. The most important one is the TaskExplorer (see Figure 3). The left panel of TaskExplorer shows the hierarchy of projects defined by the user. The right panel shows all of the resources associated with the selected project (sorted, by default, according to recency). The user can double click on any of these items to open them. Hence, the most natural way for the user to recover from an interruption is to go to TaskExplorer, select a project to resume (in the left panel), and then double-click on the relevant resources (in the right panel) to open them. A major advantage of TaskExplorer is that it pulls together all of the resources relevant to a project. In current desktop software such as Windows, the relevant resources are scattered across a variety of user interfaces including (a) email folders, (b) email contacts, (c) file system folders, (d) browser history and favorites, (e) global recent documents (in the Start menu), and (f) recent documents in each Office application. Pulling all of these resources into a single place provides a unified view of the project.

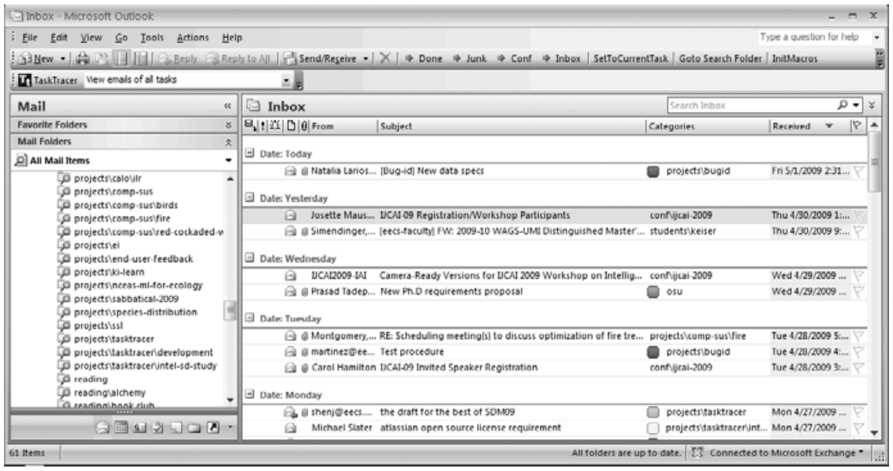

Task Explorer - Current Task: projects\tasktracer\development Selected Task: projects\tasktracer\development
Task
View
Tools
Help
All
Web
Folders
Word
Excel
PowerPoint
Mail
Contacts
Other

ki-learn
knowledge-plane
LSI
lter
mil
multi-label
nceas-ml-for-ecology
sabbatical-2009
self-aware-learning
sensor-networks
species-distribution
ssl
stats
tasktracer
development
intel-sd-study
test
tl
transfer learning
reading
alchemy
book club

Name	Path	Type	Date Modified
ep2-figures.pptx	c:\a\p\tasktracer\...	PowerPoint	5/1/2009 9:25:01 PM
email-predictor	c:\a\p\tasktracer\...	Folder	5/1/2009 9:25:01 PM
folderpredictor.log	c:\program files\ta...	File	5/1/2009 8:51:45 PM
Dietterich, Tom	/O=OREGON STA...	Contact	5/1/2009 8:51:43 PM
RE: FP bug	Dietterich, Tom</...	Outlook	5/1/2009 8:51:43 PM
Re: FP bug	Xinlong Bao<bao...	Outlook	5/1/2009 8:50:18 PM
RE: FP bug	Dietterich, Tom</...	Outlook	5/1/2009 8:11:29 PM
FolderPredictor-debug.exe	c:\downloaded-file...	File	5/1/2009 6:56:36 PM
Re: Fwd: CW for email tagging	Koby Crammer<cra...	Outlook	5/1/2009 2:33:19 PM
http://en.wikipedia.org/wiki/...	http://en.wikipedia...	IExplorer	5/1/2009 2:14:18 PM
RE: Fwd: CW for email tagging	Dietterich, Tom</...	Outlook	5/1/2009 1:23:24 PM
FW: Fwd: CW for email taggi...	Dietterich, Tom</...	Outlook	5/1/2009 1:23:00 PM
cs-tasktracer@ENGR.ORST...	cs-tasktracer@EN...	Contact	5/1/2009 1:14:38 PM
cs-tasktracer-bounces@engr...	cs-tasktracer-boun...	Contact	5/1/2009 1:14:38 PM
[Cs-tasktracer] TT3 API disc...	cs-tasktracer-boun...	Outlook	5/1/2009 1:14:38 PM
RE: job title for our undergra...	Dietterich, Tom</...	Outlook	4/28/2009 4:46:01 PM
job title for our undergrad spot	Michael Slater<slat...	Outlook	4/28/2009 4:45:12 PM

Fig. 3 The TaskExplorer displays all of the resources associated with a project, sorted by recency.

The second way that TaskTracer supports multi-tasking and interruption recovery is by making Outlook project-centered. TaskTracer implements email tagging by using the Categories feature of Outlook. This (normally hidden) feature of Outlook allows the user to define tags and associate them with email messages. The TaskTracer email tagger utlizes this mechanism by defining one category tag for each project. TaskTracer also defines a “search folder” for each tag. A search folder is another Outlook feature that is normally hidden. It allows the user to define a “view” (in the database sense) over his or her email. This view looks like a folder containing email messages (see Figure 4). In the case of TaskTracer, the view contains all email messages associated with each project. This makes it easy for the user to find relevant email messages.

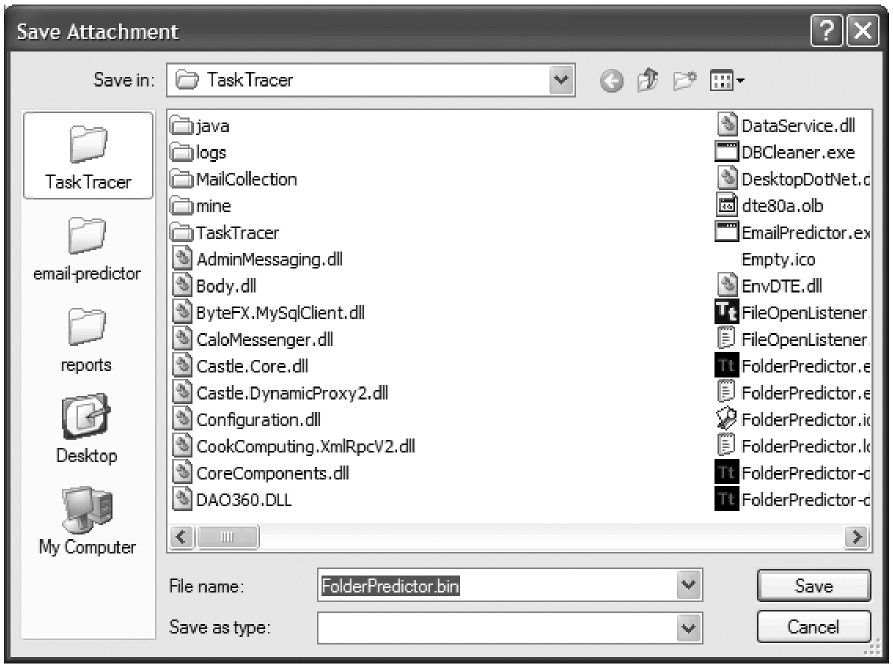

Inbox - Microsoft Outlook
File Edit View Go Tools Actions Help
Type a question for help
New Reply Reply to All Send/Receive Done Junk Conf Inbox SetToCurrentTask Goto Search Folder InitMacros
TaskTracer View emails of all tasks
Mail
Favorite Folders
Mail Folders
All Mail Items
Inbox
Search Inbox
61 Items
All folders are up to date.
Connected to Microsoft Exchange

projects\calo\ilr
projects\comp-sus
projects\comp-sus\birds
projects\comp-sus\fire
projects\comp-sus\red-cockaded-w
projects\ei
projects\end-user-feedback
projects\ki-learn
projects\nceas-ml-for-ecology
projects\sabbatical-2009
projects\species-distribution
projects\ssl
projects\tasktracer
projects\tasktracer\development
projects\tasktracer\intel-sd-study
reading
reading\alchemy
reading\book club

From	Subject	Categories	Received
Date: Today			
Natalia Larios...	[Bug-id] New data specs	projects\bugid	Fri 5/1/2009 2:31...
Date: Yesterday			
Josette Maus...	IJCAI-09 Registration/Workshop Participants	conf\ijcai-2009	Thu 4/30/2009 1:...
Simendinger,...	[eecs-faculty] FW: 2009-10 WAGS-UMI Distinguished Master'...	students\keiser	Thu 4/30/2009 9:...
Date: Wednesday			
IJCAI2009-IAI	Camera-Ready Versions for IJCAI 2009 Workshop on Intellig...	conf\ijcai-2009	Wed 4/29/2009 ...
Prasad Tadep...	New Ph.D requirements proposal	osu	Wed 4/29/2009 ...
Date: Tuesday			
Montgomery,...	RE: Scheduling meeting(s) to discuss optimization of fire tre...	projects\comp-sus\fire	Tue 4/28/2009 5:...
martinez@ee...	Test procedure	projects\bugid	Tue 4/28/2009 4:...
Carol Hamilton	IJCAI-09 Invited Speaker Registration	conf\ijcai-2009	Tue 4/28/2009 3:...
Date: Monday			
shenj@eecs....	the draft for the best of SDM09	projects\tasktracer	Mon 4/27/2009 ...
Michael Slater	atlassian open source license requirement	projects\tasktracer\int...	Mon 4/27/2009 ...

Fig. 4 TaskTracer tags each email message with the project to which it is associated. This tag is assigned as a “category” for the email message. At left are search folders for each of the TaskTracer projects.

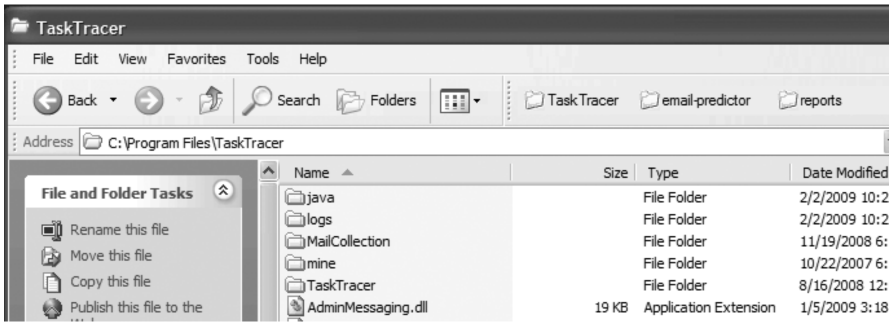
A third method of assisting the user is the Folder Predictor. TaskTracer keeps track of which folders are associated with each project. Based on the current project, it can predict which folders the user is likely to want to visit when performing an Open or Save. Before each Open/Save, TaskTracer computes a set of three folders that it believes will jointly minimize the expected number of mouse clicks to get to the target folder. It then initializes the File Open/Save dialogue box in the most likely of these three folders and places shortcuts to all three folders in the so-called “Places Bar” on the left (see Figure 5). Shortcuts to these three folders are also provided as toolbar buttons in Windows Explorer so that the user can jump directly to these folders by clicking on the buttons (see Figure 6). Our user studies have shown that these folder predictions reduce the average number of clicks by around 50% for most users as compared to the Windows default File Open/Save dialogue box.



java
logs
MailCollection
mine
TaskTracer
AdminMessaging.dll
Body.dll
ByteFX.MySqlClient.dll
CaloMessenger.dll
Castle.Core.dll
Castle.DynamicProxy2.dll
Configuration.dll
CookComputing.XmlRpcV2.dll
CoreComponents.dll
DAO360.DLL
DataService.dll
DBCleaner.exe
DesktopDotNet.c
dte80a.olb
EmailPredictor.ex
Empty.ico
EnvDTE.dll
FileOpenListener
FileOpenListener
FolderPredictor.e
FolderPredictor.e
FolderPredictor.i
FolderPredictor.l
FolderPredictor-c
FolderPredictor-c

File name: FolderPredictor.bin
Save as type:

Fig. 5 The Folder Predictor places three shortcuts in the “Places Bar” on the left of the File Open/Save dialogue box. In this case, these are “Task Tracer”, “email-predictor”, and “reports”.



C:\Program Files\TaskTracer

Name	Size	Type	Date Modified
java		File Folder	2/2/2009 10:2
logs		File Folder	2/2/2009 10:2
MailCollection		File Folder	11/19/2008 6:
mine		File Folder	10/22/2007 6:
TaskTracer		File Folder	8/16/2008 12:
AdminMessaging.dll	19 KB	Application Extension	1/5/2009 3:18

Fig. 6 The Folder Predictor adds a toolbar to the windows file browser (Windows Explorer) with the folder predictions; see top right.

2.3 *Instrumentation*

TaskTracer employs Microsoft’s addin architecture to instrument Microsoft Word, Excel, Powerpoint, Outlook, Internet Explorer, Windows Explorer, Visual Studio, and certain OS events. This instrumentation captures a wide variety of application events including the following:

- For documents: New, Open, Save, Close,
- For email messages: Open, Read, Close, Send, Open Attachment, New Email Message Arrived,
- For web pages: Open,
- For windows explorer: New folder, and
- For Windows XP OS: Window Focus, Suspend/Resume/Idle.

TaskTracer also instruments its own components to create an event whenever the user declares a change in the current project or changes the project tag(s) associated with a resource.

TaskTracer also instruments a set of provenance events that capture the flow of information *between* resources. These include the following:

- For documents: SaveAs,
- For email messages: Reply, Forward, Attach File, Save Attachment, Click on Web Hyperlink,
- For web pages: Navigate (click on hyperlink), Upload file, Download file,
- For windows explorer: Rename file, Copy file, Rename folder, and
- For Windows XP OS: Copy/Paste, Cut/Paste.

All of these events are packaged up as TaskTracer events and transmitted on the TaskTracer publish/subscribe event bus. A component called the Association Engine subscribes to these events and creates associations between resources and projects. Specifically, when a resource is opened and has window focus for at least 10 seconds, then the Association Engine automatically tags that resource with the user's current project.

3 Machine Learning for Project Associations

There are three machine learning components in TaskTracer: (a) the email tagger, (b) the project switch detector, and (c) the folder predictor. We now describe the algorithms that are employed in each of these.

3.1 The Email Tagger

When an email message arrives, TaskTracer extracts the following set of features to describe the message:

- All words in the subject line and body (with stop words removed),
- One boolean feature for each recipient email address (including the sender's email address), and
- One boolean feature for each unique *set* of recipients. This is a particularly valuable feature, because it captures the "project team".

These features are then processed by an online supervised learning algorithm to predict which project is the most appropriate to assign to this email message. Our team recently completed a comparison study of several online multiclass learning algorithms to determine which one worked best on our email data set. The data set consists of 21,000 messages received by Tom Dietterich, dating from 2004 to 2008. There are 380 classes (projects), ranging in size from a single message to 2500 messages. Spam has already been removed.

Six different machine learning algorithms were examined: Bernoulli Naive Bayes [6], Multinomial Naive Bayes [11], Transformed Weight-Normalized Complement Naive Bayes [13], Term Frequency-Inverse Document Frequency Counts [14], Online Passive Aggressive [5], and Confidence Weighted [5].

Bernoulli Naive Bayes (BNB) is the standard Naive Bayes classification algorithm which is frequently employed for simple text classification [6]. BNB estimates for each project j and each feature x, $P(x|j)$ and $P(j)$, where x is 1 if the feature (i.e., word, email address, etc.) appears in the message and 0 otherwise. A message is predicted to belong to the project j that maximizes $P(j)\prod_x P(x|j)$, where the product is taken over all possible features. (This can be implemented in time proportional to the length of the email message.)

Multinomial Naive Bayes (MNB) [11] is a variation on Bernoulli Naive Bayes in which x is a multinomial random variable that indexes the possible features, so $P(x|j)$ is a multinomial distribution. We can conceive of this as a die with one "face" for each feature. An email message is generated by first choosing the project according to $P(j)$ and then rolling the die for project j once to generate each feature in the message. A message is predicted to belong to the project j that maximizes $P(j)\prod_x P(x|j)$, but now the product x is over all appearances of a feature in the email message. Hence, multiple occurrences of the same word are captured.

Rennie et al. [13] introduced the Transformed Weight-Normalized Complement Naive Bayes (TWCNB) algorithm. This improves MNB through several small adaptations. It transforms the feature count to pull down higher counts while maintaining an identity transform on 0 and 1 counts. It uses inverse document frequency to give less weight to words common among several different projects. It normalizes word counts so that long documents do not receive too much additional weight for repeated occurrences. Instead of looking for a good match of the target email to a project, TWCNB looks for a poor match to the project's complement. It also normalizes the weights.

Term Frequency-Inverse Document Frequency (TFIDF) is a set of simple counts that reflect how closely a target email message matches a project by dividing the frequency of a feature within a project by the log of the number of times the feature appears in messages belonging to all other projects. A document is predicted to belong to the project that gives the highest sum of TFIDF counts [14].

Crammer et al. [5] introduced the Online Passive Aggressive Classifier (PA), the multiclass version of which uses TFIDF counts along with a shared set of learned weights. When an email message is correctly predicted by a large enough confidence, the weights are not changed ("passive"). When a message is incorrectly predicted, the weights are aggressively updated so that the correct project would have been predicted with a high level of confidence.

Confidence Weighted Linear Classification (CW) is an online algorithm introduced by Dredze et al. [5]. It maintains a probability distribution over the learned weights of the classifier. Similar in spirit to PA, when a prediction mistake is made, CW updates this probability distribution so that with probability greater than 0.9, the mistake would not have been made. The effect of this is to more aggressively update weights in which the classifier has less confidence and less aggressively for weights in which it has more confidence.

It should be emphasized that all of these algorithms only require time linear in the size of the input and in the number of projects. Hence, these algorithms should be considered for other cyber situation awareness settings that require high speed for learning and for making predictions.

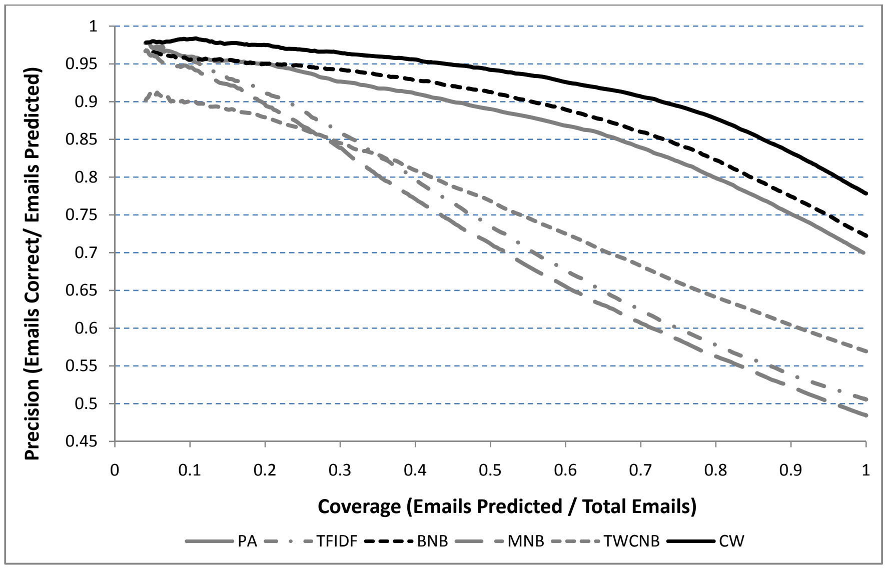


Fig. 7 Precision/Coverage curves for six online learning algorithms applied to email tagging by project.

Figure 7 shows a precision-coverage plot comparing the online performance of these algorithms. Online performance is measured by processing each email message in the order that the messages were received. First, the current classifier is applied to predict the project of the message. If the classifier predicts the wrong project, then this is counted as an error. Second, the classifier is told the correct project, and it can then learn from that information. Each curve in the graph corresponds to varying a confidence threshold θ. All of these classifiers produce a predicted score (usually a probability) for each possible project. The confidence score is the difference between the score of the top prediction and the score of the second-best prediction. If this confidence is greater than θ, then the classifier makes a prediction. Otherwise, the classifier “abstains”. By varying θ, we obtain a tradeoff between coverage (shown on the horizontal axis)—which is the percentage of email messages for which the classifier made a prediction—and precision (on the vertical axis)—which is the probability that the prediction is correct. For email, a typical user would probably want to adjust θ so that coverage is 100% and then manually correct all of the mislabeled email.

We can see that the best learning algorithm overall is the Confidence Weighted (CW) classifier. It achieves about 78% precision at 100% coverage, so the user would need to correct 22% of the email tags. In contrast, without the email tagger, the user would need to assign tags (or sort email into folders) 100% of the time, so this represents a 78% reduction in the amount of time spent tagging or sorting email messages.

One surprise is that the venerable Bernoulli Naive Bayes algorithm performed the second-best, and many classifiers that were claimed to be improvements over BNB performed substantially worse. This probably reflects the fact that email messages are quite different from ordinary textual documents.

3.2 Project Switch Detector

As we discussed above, TaskTracer monitors various desktop events (Open, Close, Save, SaveAs, Change Window Focus, and so on). In addition, once every minute (or when the user declares a project switch), TaskTracer 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: $\mathbf{F}_P$:$(\mathbf{X}_t, y_j) \rightarrow \mathbb{R}^k$ and $\mathbf{F}_S : (\mathbf{X}_t) \rightarrow \mathbb{R}^m$. The first function $\mathbf{F}_P$ computes *project-specific* features for a specified project y_j; the second function $\mathbf{F}_S$ computes *switch-specific* features. The project-specific features include

- Strength of association of the active resource with project 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 project 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 project and then the resource is visited), this is a weaker indication that the current project is y_j.
- Percentage of open resources associated with project y_j: if most open resources are associated with y_j, it is likely that y_j is the current project.
- Importance of window title word x to project y_j. Given the bag of words Ω, we compute a variant of TF-IDF [8] for each word x and project y_j: $\mathrm{TF}(x,\Omega) \cdot \log \frac{|\overline{S}|}{\mathrm{DF}(x,\overline{S})}$. Here, $\overline{S}$ is the set of all feature vectors not labeled as y_j, $\mathrm{TF}(x,\Omega)$ is the number of times x appears in Ω and $\mathrm{DF}(x,\overline{S})$ is the number of feature vectors containing x that are not labeled y_j.

These project-specific features are intended to predict whether y_j is the current project.

The switch-specific features predict the likelihood of a switch regardless of which projects are involved. They include

- Number of resources closed in the last 60 seconds: if the user is switching projects, many open resources will often be closed.
- Percentage of open resources that have been accessed in the last 60 seconds: if the user is still actively accessing open resources, it is unlikely there is a project switch.
- The time since the user's last explicit project 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 project 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 projects for time intervals $t-1$ and t. Given a project pair $\langle y_{t-1}, y_t\rangle$, the scoring function g is defined as

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

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 projects at time $t-1$ and t (respectively). The third term measures the likelihood that there is no project switch from time $t-1$ to t. Thus, the third component of g serves as a "switch penalty" when $y_{t-1} \neq y_t$.

The project switch detector searches for the pair $\langle \hat{y}_1, \hat{y}_2\rangle$ that maximizes the score function g. If $\hat{y}_2$ is different from the current declared project and the score is larger than a confidence threshold, then a switch is predicted. At first glance, this search over all pairs of projects would appear to require time quadratic in the number of projects. However, the following algorithm computes the best score in linear time:

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

Each pair of lines can be computed in time linear in the number of projects. 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 project 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 projects at time $t-1$ and t.

A desirable aspect of this formulation is that the classifier is still linear in the weights Λ. Hence, we can apply any learning algorithm for linear classification to this problem. We chose to apply a modified version of the Passive-Aggressive (PA) algorithm discussed above. The standard Passive-Aggressive algorithm works as follows: Let the real projects 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 project pair. When the system makes an error, PA updates Λ by solving the following constrained optimization problem:

$$\begin{aligned} \Lambda_{t+1} = \operatorname*{argmin}_{\Lambda \in \mathbb{R}^n} & \frac{1}{2} \|\Lambda - \Lambda_t\|_2^2 + C\xi^2 \\ \text{subject to } & g(\langle y_1, y_2 \rangle) - g(\langle \hat{y}_1, \hat{y}_2 \rangle) \geq 1 - \xi . \end{aligned}$$

The first term of the objective function, $\frac{1}{2} \|\Lambda - \Lambda_t\|_2^2$, says that Λ should change as little as possible (in Euclidean distance) from its current value Λ_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 project pair should be larger than the score of the incorrect project 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 ξ 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 projects. In any case, the second term in the objective function, $C\xi^2$, serves to encourage ξ to be small. The constant parameter C controls the tradeoff between taking small steps (the first term) and fitting the data (driving ξ to zero). Crammer et al. [2] 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 Λ 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 Λ. The modified weight-update optimization problem is the following:

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

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

As with Passive-Aggressive, this optimization problem has a closed-form solution. Define $\mathbf{Z}_t = \left\langle \mathbf{Z}_t^1, \mathbf{Z}_t^2, \mathbf{Z}_t^3 \right\rangle$ where

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

Then the updated weight vector can be computed as

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

where

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

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.

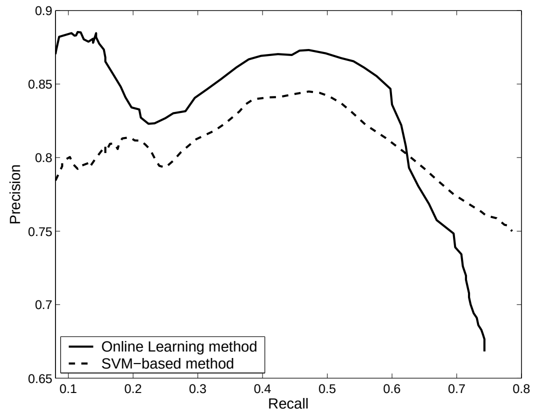


Fig. 8 User 1: Precision of different learning methods as a function of the recall, created by varying the confidence threshold.

To evaluate the Project Switch Detector, 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 earlier version of the Switch Detector that employed a simpler set of features and a support vector machine (SVM) classifier was running throughout this time, and the users tried to provide feedback to the system throughout.

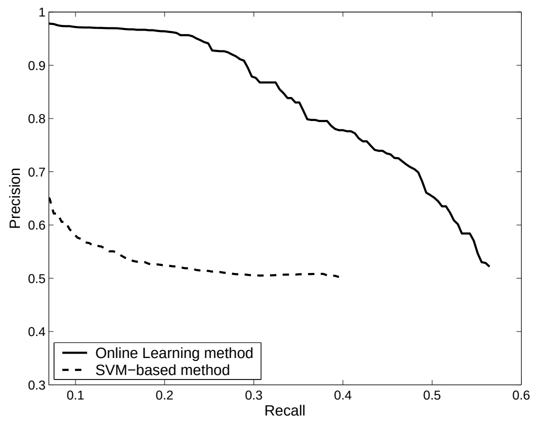


Fig. 9 User 2: Precision of different learning methods as a function of the recall, created by varying the confidence threshold.

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

To evaluate the online learning algorithm, we make the assumption that project switches observed in the users' data are all correct. We then 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 project 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 project 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 separate 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 the SVM approach are plotted in Figures 8 and 9. The SVM method only uses the bag of words from the window titles and pathname/URL to predict project switches. As we described above, the online passive-aggressive approach incorporates much richer contextual information. This makes our new approach more accurate. For Figure 8, we see that for levels of recall below 60%, the passive-aggressive approach has higher precision. If we tune the confidence threshold to achieve 50% recall (so half of all project switches are missed), the precision is greater than 85%, so there are only 15% false predictions. Qualitatively, the user reports that these false predictions are typically very sensible. For example, suppose the user is working on a new project P_{new} and needs to access a document from a previous project P_{old}. When the document is opened, the project switch detector will predict that the user is switching to P_{old}, but in fact, the user wants the document to now become associated with P_{new}. It is hard to see how the project switch detector can avoid this kind of error. For the second user, the online passive-aggressive approach is hugely superior to the SVM method.

It should be noted that the new method is also much more efficient than the SVM method. On an ordinary PC, the passive-aggressive algorithm only took 4 minutes to make predictions for User 1's 65,049 instances while the SVM approach needed more than 12 hours!

3.3 The Folder Predictor

The Folder Predictor is much simpler than either of the other two learning methods. Folder Predictor maintains a count $N(j,f)$ for each project j and folder f. $N(j,f)$ is the discounted number of opens or saves of files stored in folder f while the current project is j. Each time the user opens or saves a file in folder f when the current project is j, this count is updated as

$$N(j,f) := \rho N(j,f) + 1,$$

where ρ is a discount factor, which we typically set at 0.85. At the same time, for all other folders, $f' \neq f$, the counts are updated as

$$N(j,f') := \rho N(j,f').$$

Given these statistics, when the user initiates a new Open or Save, Folder Predictor estimates the probability that the user will want to use folder f as

$$P(f|j) = \frac{N(j,f)}{\sum_f N(j,f)},$$

where j is the user's current project.

As we described in Section 2, the Folder Predictor modifies the "Places Bar" of the File Open/Save dialogue box to include shortcuts to three folders, which we will refer to as f_1 (the top-most), f_2, and f_3. In addition, where Windows permits, TaskTracer initializes the dialogue box to start in f_1. Given the probability distribution $P(f|j)$, the Folder Predictor chooses these three folders in order to minimize the expected number of clicks that the user will need to perform in order to reach the user's desired folder. Specifically, Folder Predictor computes (f_1,f_2,f_3) as follows:

$$(f_1,f_2,f_3) = \operatorname*{argmin}_{(f_1,f_2,f_3)} \sum_f P(f|j) \min\{clicks(f_1,f),1+clicks(f_2,f),1+clicks(f_3,f)\}. \tag{1}$$

In this formula, $clicks(i, j)$ is the number of clicks (or double clicks) it takes to navigate from folder i to folder j in the folder hierarchy. We performed a user study that found, among other things, that the most common way that users find and open files is by clicking up and down in the folder hierarchy using the File Open/Save dialogue box. Hence, the number of click operations (were a single click or a double click are both counted as one operation) is just the tree distance between folders i and j. The min in Equation 1 assumes that the user remembers enough about the layout of the folder hierarchy to know which of the three folders f_1, f_2, or f_3 would be the best starting point for navigating to the target folder. The second and third items inside this minimization include an extra click operation, because the user must click on the Places Bar shortcut before navigating through the hierarchy.

Users can have thousands of folders, so we do not want to consider every possible folder as a candidate for f_1, f_2, and f_3. Instead, we consider only folders for which $P(f|j) > 0$ and the ancestors of those folders in the hierarchy.

Table 1 Folder Predictor Data Sets

ID	User Type	Data Collection Time	Set Size
1	Professor	12 months	1748
2	Professor	4 months	506
3	Graduate Student	7 months	577
4	Graduate Student	6 months	397

To evaluate the Folder Predictor, we collected Open/Save data from four TaskTracer users. Table 1 summarizes the data. One beautiful aspect of folder prediction is that after making a prediction, TaskTracer will always observe the user's true target folder, so there is no need for the user to provide any special feedback to the Folder Predictor. We added a small amount of instrumentation to collect the folder that Windows would have used if Folder Predictor had not been running.

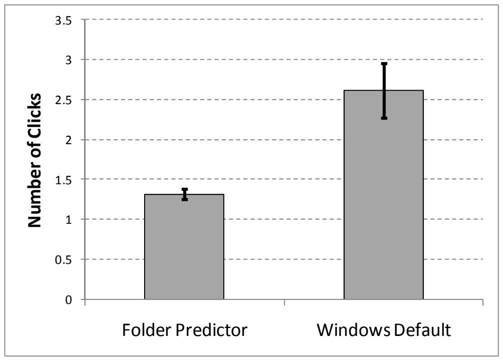


Fig. 10 Mean number of clicks to reach the target folder

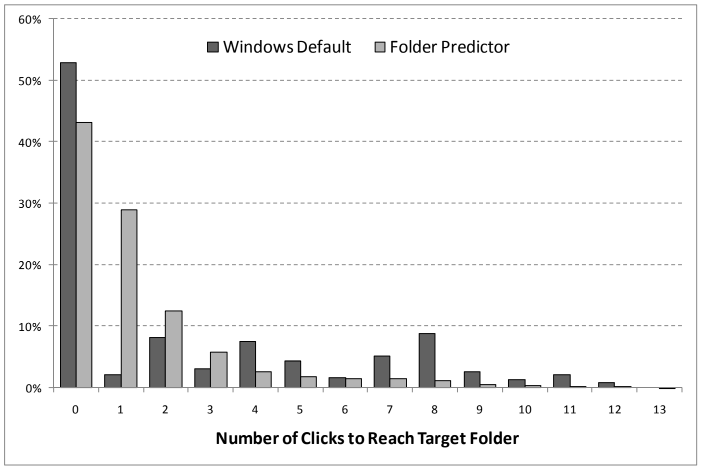


Fig. 11 Histogram of the number of clicks required to reach the user's target folder.

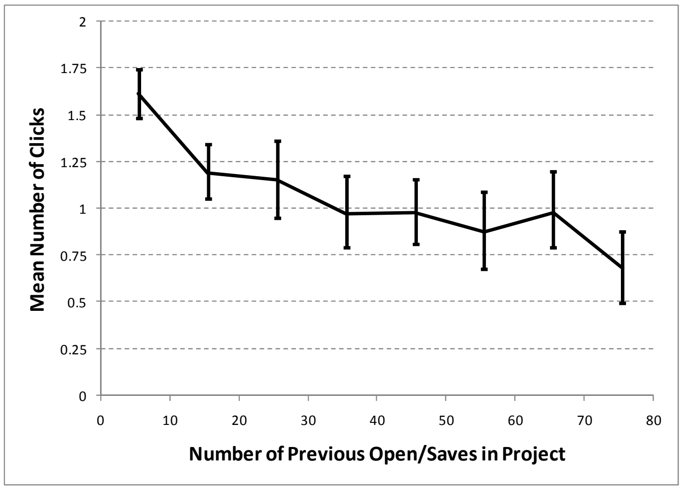


Fig. 12 Learning curve for the Folder Predictor showing the number of clicks as a function of the amount of training data (opens and saves) within the project.

Figure 10 compares the average number of clicks for these four users using Folder Predictor with the average number of clicks that would have been required by the Windows defaults. Folder Predictor has reduced the number of clicks by 49.9%. The reduction is statistically significant ($p < 10^{-28}$). Figure 11 shows a histogram of the number of clicks required to reach the target folder under the windows default and the folder predictor. Here we see that the Windows default starts in the target folder more than 50% of the time, whereas Folder Predictor only does this 42% of the time. But if the Windows default is wrong, then the target folder is rarely less than 2 clicks away and often 7, 8, or even 12 clicks away. In contrast, the Folder Predictor is often one click away, and the histogram falls smoothly and rapidly. The reason Folder Predictor is often one click away is that if $P(f|j)$ is positive for two or more sibling folders, the folder the minimizes the expected number of clicks is often the parent of those folders. The parent is only 1 click away, whereas if we predict the wrong sibling, then the other sibling is 2 clicks away.

Figure 12 shows a learning curve for Folder Predictor. We see that as it observes more Opens and Saves, it becomes more accurate. After 30 Opens/Saves, the expected number of clicks is less than 1.

Qualitatively, Folder Predictor tends to predict folders that are widely spaced in the folder hierarchy. Users of TaskTracer report that it is their favorite feature of the system.

4 Discovering User Workflows

For many knowledge workers, a substantial fraction of their time at the computer desktop is occupied with routine workflows such as writing reports, performance reviews, filling out travel reimbursements, and so on. It is easy for knowledge workers to lose their situation awareness and forget to complete a workflow. This is true even when people maintain “to-do” lists (paper or electronic). Indeed, electronic to-do managers have generally failed to help users maintain situation awareness. One potential reason is that these tools introduce substantial additional work, because the user must not only execute the workflows but also maintain the to-do list [1].

An interesting challenge for high level situation awareness is to create a to-do manager that can maintain itself. Such a to-do manager would need to detect when a new to-do item should be created and when the to-do item is completed. More generally, an intelligent to-do manager should track the status of each to-do item (due date, percentage complete, etc.). Often, a to-do item requires making requests for other people to perform certain steps, so it would be important for a to-do manager to keep track of items that are blocked waiting for other people (and offer to send reminder emails as appropriate).

A prerequisite for creating such an intelligent to-do manager is developing a system that can discover and track the workflows of desktop knowledge workers. The central technical challenge is that because of multi-tasking, desktop workflows are interleaved with vast amounts of irrelevant activity. For example, a workflow for assembling a quarterly report might require two weeks from start to finish. During those two weeks, a busy manager might receive 1000 email messages, visit several hundred web pages, work on 50-100 documents, and make progress on dozens of other workflows. How can we discover workflows and then track them when they are embedded in unrelated multi-tasking activity?

Our solution to this conundrum is to assume that a workflow will be a connected subgraph within an *information flow graph* that captures flows of information among resources. For example, a travel authorization workflow might involve first exchanging email messages with a travel agent and then pasting the travel details into a Word form and emailing it to the travel office. Finally, the travel office replies with an authorization code. Figure 13 shows a schematic representation of this workflow as a directed graph. The advantage of this connected-subgraph approach is that it allows us to completely ignore all other desktop activity and focus only on those events that are connected to one another via information flows. Of course the risk of this approach is that in order for it to succeed, we must have *complete* coverage of all possible information flows. If an information flow link is missed, then the workflow is no longer a connected graph, and we will not be able to discover or track it.

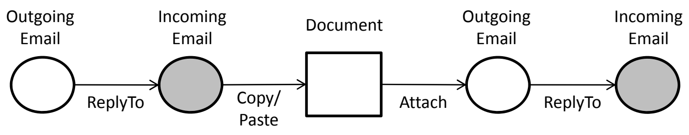


Fig. 13 Information flow graph for the travel authorization workflow. Circular nodes denote email messages, and squares denote documents. Shaded nodes denote incoming email messages. Each node is labeled with the type of resource, and each edge is labeled with an information flow action. Multiple edges can connect the same two nodes (e.g., SaveAs and Copy/Paste).

Our current instrumentation in TaskTracer captures many, but not all, important information flows. Cases that we do not capture include pasting into web forms, printing a document as a pdf file, exchanging files via USB drives, and utilities that zip and unzip collections of files. We are also not able to track information flows that occur through “cloud” computing tools such as Gmail, GoogleDocs, SharePoint, and wiki tools. Finally, sometimes the user copies information visually (by looking at one window and typing in another), and we do not detect this information flow. Nonetheless, our provenance instrumentation does capture enough information flows to allow us to test the feasibility of the information-graph approach.

The remainder of this section describes our initial efforts in this direction. These consist of three main steps: (a) building the information flow graph, (b) mining the information flow graph to discover workflows, and (c) applying an existing system, WARP, to track these workflows.

4.1 Building the Information Flow Graph

The basic information flow graph is constructed from the provenance events captured by TaskTracer. In addition, for this work, we manually added two other information flow events that are not currently captured by our instrumentation: (a) converting from Word files to PDF files and (b) referring to a document from an email message (e.g., by mentioning the document title or matching keywords).

As we will see below, our graph mining algorithm cannot discover loops. Nonetheless, we can handle certain kinds of simple loops by pre-processing the graph. Specifically, if the graph contains a sequence of SaveAs links (e.g., because the user created a sequence of versions of a document), this sequence is collapsed to a single **SaveAs*** relationship. Similarly, if the graph contains a sequence of email messages to and from a single email address, this is collapsed to a single **ReplyTo*** relationship.

4.2 Mining the Information Flow Graph

The goal of our graph mining algorithm is to find all frequent subgraphs of the information flow graph. These correspond to recurring workflows. Two subgraphs match if the types of the resources match and if they have the same set of edges with the same event labels. A subgraph is frequent if it occurs more than s times; s is called the minimum support threshold.

To find frequent subgraphs, we apply a two step process. The first step is to find frequent subgraphs while ignoring the labels on the edges. We apply the GASTON algorithm of Nijssen and Kok [12] to find maximal subgraphs that appear at least s times in the information flow graph. The second step is then to add edge labels to these frequent subgraphs to find all frequent labeled subgraphs. We developed a dynamic programming algorithm that can efficiently find these most frequent labeled subgraphs [15].

4.3 Recognizing Workflows

The labeled subgraphs can be converted to a formalism called the Logical Hidden Markov Model or LHMM [10]. An LHMM is like a standard HMM except that the states are logical atoms such as ATTACH(MSG1, FILE1), which denotes the action of attaching a file (FILE1) to an email message (MSG1). This formalism allows us to represent the parameters of a workflow, such as the sender and recipients of an email message, the name of a file, and so on.

Each frequent labeled subgraph is converted to a LHMM as follows. First, each edge in the subgraph becomes a state in the LHMM. This state generates the corresponding observation (the TaskTracer event) with probability 1. An initial "Begin" state and a final “End” state are appended to the start and end of the LHMM. Second, state transitions are added to the LHMM for each observed transition in the matching subgraphs from the information flow graph. If an edge was one of the “loop” edges (ReplyTo* or SaveAs*), then a self-loop transition is added to the LHMM. Transition probabilities are estimated based on the observed number of transitions in the information flow graph. The “Begin” state is given a self-loop with probability α. Finally, each state transition in the LHMM is replaced by a transition to a unique “Background” state. The Background state generates *any* observable event. It also has a self-loop with probability β. This state is intended to represent all of the user actions that are not part of the workflow.

Hung Bui (personal communication) has developed a system called WARP that implements a recognizer for logical hidden Markov networks. It does this by performing a “lifted” version of the usual probabilistic reasoning algorithms designed for standard HMMs. WARP handles multiple interleaved workflows by using a Rao-Blackwellized Particle Filter [4], which is an efficient approximate inference algorithm.

4.4 Experimental Evaluation

To evaluate our discovery method, we collected desktop data from four participants at SRI International as part of the DARPA-sponsored CALO project. The participants performed a variety of workflows involving preparing, submitting, and reviewing conference papers, preparing and submitting quarterly reports, and submitting travel reimbursements. These were interleaved with other routine activities (reading online newspapers, handling email correspondence). Events were captured via TaskTracer instrumentation and augmented with the two additional event types discussed above to create information flow graphs. Figure 14 shows one of the four flow graphs. There were 26 instances of known workflows in the four resulting information flow graphs.

Figure 15 shows an example of one discovered workflow. This workflow arose as part of two scenarios: (a) preparing a quarterly report and (b) preparing a conference paper. In both cases, an email message is received with an attached file. For the quarterly report, this file is a report template. For the conference paper, the file as a draft paper from a coauthor. The user saves the attachment and then edits the file through one or more SaveAs events. Finally, the user attaches the edited file to an email that is a reply to the original email message.

Another workflow that we discovered was the counterpart to this in which the user starts with a document, attaches it to an outgoing email message, and sends it to another person (e.g., to have them edit the document and return it). There is a series of exchanged emails leading to an email reply with an attached file from which the user saves the attachment. Our current system is not able to fuse these two workflows into a multi-user workflow, although that is an important direction for future research.

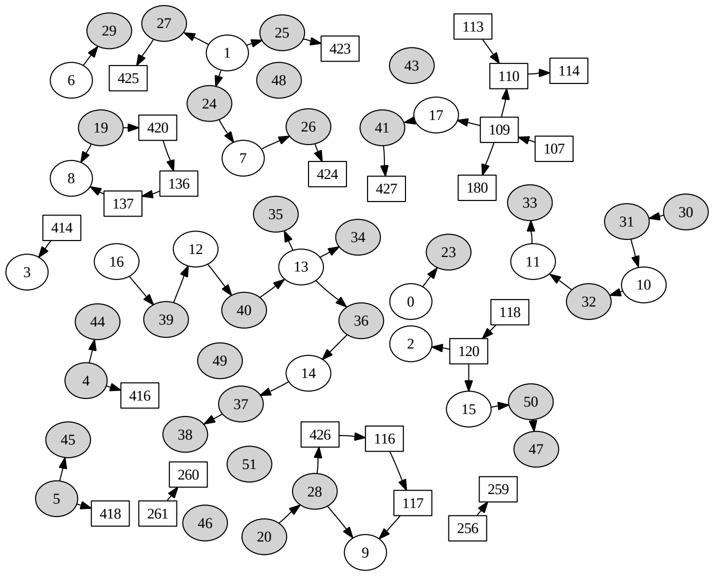


Fig. 14 Example of an information flow graph. Round nodes are email messages and rectangles are documents. Shaded nodes are incoming email messages. Each node is numbered with its resource id number.

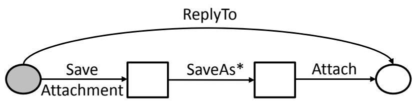


Fig. 15 Example of a discovered workflow.

Three other workflows or workflow sub-procedures were discovered from the four participant's information graphs.

To evaluate the method, we performed two experiments. First, we conducted a leave-one-user-out cross-validation by computing how well the workflows discovered using the three remaining users matched the information flow graph of the held-out user. For each information graph, we first manually identified all instances of known workflows. Then for each case where a discovered workflow matched a subgraph of the information graph, we scored the match by whether it overlapped a known workflow. We computed the precision, recall, and F1 score of the matched nodes and arcs for the true workflows. The precision is the fraction of the matched nodes and arcs in the information graph that are part of known workflows. The recall is the fraction of all nodes and arcs in known workflows that are matched by some discovered workflow subgraph. The F1 score is computed as

$$F1 = \frac{2(\text{precision} \times \text{recall})}{\text{precision} + \text{recall}}.$$

With a minimum support of 3, we achieved an F1 score of 91%, which is nearly perfect.

The second experiment was to test the ability of the WARP engine to recognize these workflows in real time. Each discovered workflow was converted to a Logical HMM. Then the events recorded from the four participants were replayed in time order and processed by WARP. For each event, WARP must decide whether it is the next event of an active workflow instance, the beginning of a new workflow instance, or a background event. After each event, we scored whether WARP had correctly interpreted the event. If it had not, we then corrected WARP and continued the processing. We computed WARP's precision and recall as follows. Precision is the ratio of the number of workflow states correctly recognized divided by the total number recognized. Recall is the ratio of the number of workflow states correctly recognized divided by the total number of true workflow states in the event sequence. WARP obtained a precision of 91.3%, a recall of 66.7%, and an F1 score of 77.1%. As these statistics show, WARP is failing to detect many states. Most of these were the initial states of workflows. An analysis of these cases shows that most initial states involve an email message. There are many email messages, but only a small fraction of them initiate new workflows. We believe that distinguishing these "workflow initiation" emails requires analyzing the body of the email to detect information such as a request for comments, a call for papers, a response to a previous request, and so on. Neither TaskTracer nor WARP currently has any ability to do this kind of analysis.

5 Discussion

This chapter has presented the TaskTracer system and its machine learning components as well as a workflow discovery system and its workflow tracking capabilities. These operate at two different levels of abstraction. TaskTracer is tracking high level projects. These projects are unstructured tags associated with a set of resources (files, folders, email messages, web pages, and email addresses). TaskTracer does a very good job of tracking these high level projects, and it is able to use this project information to organize the user's information and support interruption recovery and information re-finding.

The workflow discovery work is looking for more detailed patterns of information flow. The most important idea in this work is to capture a large set of information flow actions, represent those as an information flow graph, and then formulate the workflow discovery problem as one of finding frequently-occurring subgraphs within the information flow graph. The workflow discovery algorithm was able to find the key subgraphs corresponding to known workflows. However, it often did not discover the complete workflows but only the "kernels". For example, a complete workflow for quarterly reporting involved a combination of two workflow fragments discovered by our system: one fragment for receiving the report template and filling it out and another fragment for sending the report template to multiple team members, collecting their responses, and then editing them into the original report template. Hence, the workflows discovered by the system do not necessarily correspond directly to the user's notion of a workflow. Nonetheless, these discovered workflows could be very useful for providing a kind of "auto-completion" capability, where the system could offer to perform certain steps in a workflow (e.g., saving a file attachment or initiating an email reply with the relevant file attached).

Another shortcoming of our workflow discovery approach is that it cannot discover workflows involving conditional (or unconditional) branching. Unconditional branching occurs when there are multiple ways of achieving the same goals. For example, when submitting a letter of reference, a web page may support either pasting text into a text box or uploading a text file to a web site. Conditional branching could occur when a workflow involves different steps under certain conditions. For example, a travel requisition may require different steps for international travel versus domestic travel. Extending our workflow discovery methods to handle such cases is an important area for future research.

The focus of this chapter has been on high level situation awareness to support the knowledge worker. But it is also interesting to consider ways in which high level situation awareness could help with cyber defense. One possibility is that this high level situation awareness could provide greater contextual understanding of lower-level actions. For example, if a user is visiting a novel web page or sending email to a novel recipient, this could be the start of an insider attack or it could be a new instance of a known workflow. In the latter case, it is unlikely to be an attack.

A second possibility is that this higher level situation awareness could help distinguish those actions (e.g., sending email messages, accessing web pages) that were intentionally initiated by the user from actions initiated by malware. The user-initiated actions should involve known workflows and known projects, whereas the malware actions would be more likely to involve files and web sites unrelated to the user's current project.

6 Concluding Remarks

The difficulty of maintaining cyber situation awareness varies depending on the level of abstraction that is required. This chapter has described two of the highest levels: projects and workflows. An important challenge for future research is to integrate situation awareness at all levels to provide systems that are able to exploit a broad range of instrumentation and context to achieve high accuracy, rapid response, and very low false alarm rates. Research is still quite far from this goal, but the work reported in this chapter suggests that this goal is ultimately achievable.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DARPA or the Air Force Research Laboratory (AFRL).

References

1. Bellotti, V., Dalal, B., Good, N., Bobrow, D.G., Ducheneaut, N.: What a to-do: studies of task management towards the design of a personal task list manager. In: ACM Conference on Human Factors in Computing Systems (CHI2004), pp. 735–742. ACM, NY (2004)
2. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. Journal of Machine Learning Research **7**, 551–585 (2006)
3. Dietterich, T.G., Slater, M., Bao, X., Cao, J., Lonsdale, H., Spence, C., Hadley, G., Wynn, E.: Quantifying and supporting multitasking for intel knowledge workers. Tech. rep., Oregon State University, School of EECS (2009)
4. Doucet, A., de Freitas, N., Murphy, K.P., Russell, S.J.: Rao-Blackwellised particle filtering for dynamic Bayesian networks. In: UAI'00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, pp. 176–183. Morgan Kaufmann (2000)
5. Dredze, M., Crammer, K., Pereira, F.: Confidence-weighted linear classification. In: A. McCallum, S. Roweis (eds.) Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008), pp. 264–271. Omnipress (2008)
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, Second Edition. John Wiley and Sons, Inc. (2000)
7. Gonzalez, V.M., Mark, G.: "constant, constant, multi-tasking craziness": Managing multiple working spheres. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 113–120. ACM Press (2004)
8. Joachims, T.: Transductive inference for text classification using support vector machines. In: Proceedings of the 16th International Conference on Machine Learning (ICML), pp. 200–209. Morgan Kaufmann, Bled, Slovenia (1999)
9. Kaptelinin, V.: UMEA: Translating interaction histories into project contexts. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 353–360. ACM Press (2003)
10. Kersting, K., De Raedt, L., Raiko, T.: Logial hidden Markov models. Journal of Artificial Intelligence Research (JAIR) **25**, 425–456 (2006)
11. McCallum, A., Nigam, K.: A comparison of event models for naive Bayes text classification. In: AAAI-98 Workshop on Learning for Text Categorization (1998)
12. Nijssen, S., Kok, J.N.: A quickstart in frequent structure mining can make a difference. In: Proceedings of KDD-2004, pp. 647–652 (2004)
13. Rennie, J.D.M., Shih, L., Teevan, J., R., K.D.: Tackling the poor assumptions of naive Bayes text classifiers. In: Proceedings of the International Conference on Machine Learning (ICML2003), pp. 616–623 (2003)
14. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. In: Information Processing and Management, pp. 513–523 (1988)
15. Shen, J., Fitzhenry, E., Dietterich, T.: Discovering frequent work procedures from resource connections. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI-2009), pp. 277–286. ACM, New York, NY (2009)