

# Predicting User Tasks: I Know What You're Doing!

Simone Stumpf, Xinlong Bao, Anton Dragunov, Thomas G. Dietterich, Jon Herlocker, Kevin Johnsruide, Lida Li, JianQiang Shen

School of Electrical Engineering  
Oregon State University  
Corvallis, OR  
stumpf@eecs.oregonstate.edu

## *Abstract*

Knowledge workers spend the majority of their working hours processing and manipulating information. These users face continual costs as they switch between tasks to retrieve and create information. The TaskTracer project at Oregon State University is investigating the possibilities of a desktop software system that will record in detail how knowledge workers complete tasks, and intelligently leverage that information to increase efficiency and productivity. Our approach combines human-computer interaction and machine learning to assign each observed action (opening a file, saving a file, sending an email, cutting and pasting information, etc.) to a task for which it is likely being performed. In this paper we report on ways we have applied machine learning in this environment and lessons learned so far.

## **Introduction**

Knowledge workers spend the majority of their working hours processing and manipulating information. These users face continual costs as they switch between tasks to retrieve and create information. The information may be encoded in many different formats: documents, software code, web pages, email messages, phone conversations. The cost to the user of finding information may be cognitive: workers may have to remember exactly where they were in a chain of logic, or why they decided to take their most recent action on a task. The cost may also lie in the manual interaction needed to access the necessary resources (e.g., documents and/or software tools).

Knowledge workers organize their work into discrete and describable units, such as projects, tasks or to-do items. The TaskTracer project at Oregon State University is investigating the possibilities of a desktop software system that will record in detail how knowledge workers complete tasks, and intelligently leverage that information to increase efficiency and productivity. Our goal is to develop five capabilities: more task-aware user interfaces in the applications we use daily, more efficient task-interruption recovery, better personal information management, workgroup information management and within-workgroup workflow detection and analysis. Our system operates in the Microsoft Windows environment, tracking most interactions with desktop applications as well as tracking phone calls. Our approach combines

human-computer interaction and machine learning to assign each observed action (opening a file, saving a file, sending an email, cutting and pasting information, etc.) to a task for which it is likely being performed. Once we have the past actions structured by task, we can provide substantial value to the knowledge worker in assisting in their daily task routines.

There is a substantial set of research challenges that must be faced in order to successfully develop the TaskTracer system with these capabilities. These challenges include user interface design, machine learning, privacy and workplace culture, data collection, systems architecture, and data modeling. In this paper we report on ways we have applied machine learning in this environment and lessons learned so far.

## **Task-tracking and task-related systems**

There have been previous efforts to build environments that enable knowledge workers to manage multiple concurrent activities, which we call tasks, and use knowledge of those activities to improve productivity. Workspaces (Bannon et al. 1983) can define tasks that comprise information resources (usually documents and tools for their processing) that are necessary to accomplish the goal associated with the task. Some systems work on the idea of physically separating tasks by requiring users to create project-specific folders, or set up a virtual desktop for each particular task (Card and Henderson 1987, Robertson et al. 2000). Other systems work at a more abstract level by organizing task-specific workspaces using “filters” applied to communication threads (Bellotti et al. 2003), streams or networks of documents (Freeman and Gelernter 1996, Dourish et al. 1999).

To be of assistance to a user, an agent (whether it is a computer system or a human associate) must “know” what the user is currently doing. In addition to the resources used in a task, it also seems reasonable to record users’ actions performed on those resources. The rationale behind this is that to have the correct comprehension of the task context for some resources we must consider in which way and for what reason they were accessed. For instance, the same document (say, a text file) may be opened for two completely different purposes: 1) for reading and 2) for authoring. Various systems (Fenstermacher and Ginsburg

2002, Kaptelinin 2003, Canny 2004) address this issue by aiming at recording as much information as possible about users' activities when they interact with computers. These activity records are obtained via monitoring the computer file system, input devices, and applications.

Our software, TaskTracer, employs an extensive data-collection framework to obtain detailed observations of user interactions in the common productivity applications used in knowledge work (Dragunov et al. 2004). Currently, events are collected from Microsoft Office 2003, Microsoft Visual Studio .NET, Windows XP operating system and phone calls. In this framework, TaskTracer collects file pathnames for file create, change, open, print and save, text selection, copy-paste, windows focus, web navigation, phone call, clipboard and email events. Phone call data collection uses Caller Id to collect names and phone numbers of callers. In addition, speech-to-text software collects the user's — but not the caller's — phone speech. All events are captured as individual EventMessages which contain:

- *Type*: Event type. For example, TaskTracer captures window focus, file open, file save, web page navigation, text selection, and many other events on both the applications and the operating system levels.
- *Window ID*: Window handle for windows, zero otherwise.
- *Listener Version*: Changes every time we change or add to the EventMessages the Listener can send and process. This allows backward compatibility as we change our data capture.
- *Listener ID*, the source of the EventMessage: MS Office pro-grams, file system hooks, user, clipboard, phone, etc.
- *Body Type, Body*: Event or document data in XML format.
- *Time*: Time the event fired.

Instead of using unsupervised clustering to discover tasks (Canny 2004), users of TaskTracer manually specify what tasks they are doing in the initial stage of data collection, so that each action of the user (a User Interface event) will be tagged with a particular task identifier to train predictors. We believe that we can learn to reliably predict the users' current task and task switches, and thus we can create complete and detailed records of what has been done on every task (past and present). All EventMessages are stored in a database in raw form so that researchers can analyze the history of user events. A variety of learning models can be tested on identical data sets. We are currently researching learning models based on the event data for predicting the current task of the user, for detecting when the user has changed tasks and for reducing the cost of accessing resources whilst carrying out a task.

## Plan Recognition and Task Prediction

There is a range of plan recognition tasks that people have addressed (Ourston and Mooney 1990, Davison and Hirsh 1998, Bauer 1999). For example, some work has been carried out to recognize that someone is executing an instance of a particular plan and suggest the next action. A plan often has flexibility but the user is executing a specific structured activity (e.g., taking money out of an ATM, calibrating a glucose meter).

What we are addressing is supporting an unstructured activity (e.g. writing the AAAI submission, putting together a research study). These activities typically have no or only a loosely fixed structure and are highly distinctive to the individual knowledge worker. Hence, the way that we use the term "task" is a user-defined concept name, instead of a sequence of user actions (as an aside, we would call this sequence an "event stream").

These two approaches vary mostly in their degree of sequential/hierarchical structure in the activity. We are not suggesting that these two approaches are mutually exclusive, indeed, much can be learned from plan recognition.

It could be argued that no effective user support is possible without a deep structure that can be used to explain the observed user behavior. However, we are not trying to explain the user behavior itself since the user is very competent already in deciding what to do (and what to do next). What we are trying to achieve instead is the reduction of the costs that knowledge workers face when they carry out their tasks by keeping task-related information organized. Costs may be physical/mechanical such as the number of user interface interactions (mouse, keyboard, etc) needed to achieve a goal. Costs may sometimes be in time. There are also cognitive costs, such as the remembering where a piece of information was filed or learning any new features. In addition to the "actual" costs that workers encounter while pursuing their tasks, we must also be particularly aware of the perceived costs of using any features.

## TaskPredictor and FolderPredictor

Automatic translation of interaction histories into project contexts is very challenging to implement (Kaptelinin 2003). If users must indicate task switching manually (as currently implemented in TaskTracer), this will create additional cognitive and physical costs for users, since they will have to 1) mentally structure their activities and 2) perform additional actions not directly related to the current goals — select tasks from lists, type in task titles and descriptions, etc. We believe that we can reduce these costs by combining probabilistic machine learning approaches with appropriate user interfaces that maximize online learning whilst reducing the cost on the user.

There are three main challenges to the machine learning approach. Firstly, accuracy must be exceptionally high to be acceptable to the user. Secondly, manual task switches

have fuzzy boundaries. For example, if a user has finished editing a document on a task and wants to edit another resource on a different task, making a manual task switch may introduce noise: should it be changed while they are still viewing the current document but before launching the new one or after closing the old one and opening the new one? Secondly, users may achieve the same task in different ways, hence doing something on the same task can generate different event streams. Conversely, different tasks may utilize the same objects, i.e. events and resources.

TaskPredictor is a component in TaskTracer that predicts the currently active task and sets the current task to the predicted task on the user's behalf, thereby reducing the cost to the user to switch tasks explicitly (see Figure 1, a & b). Online learning is utilized to update the model if the user corrects the predicted task.

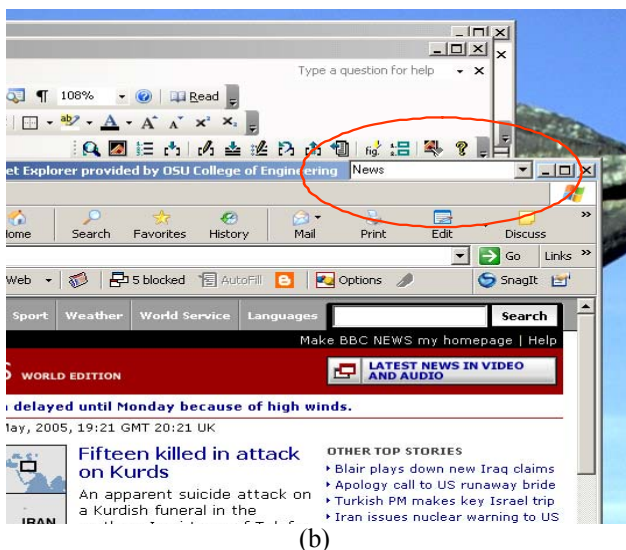
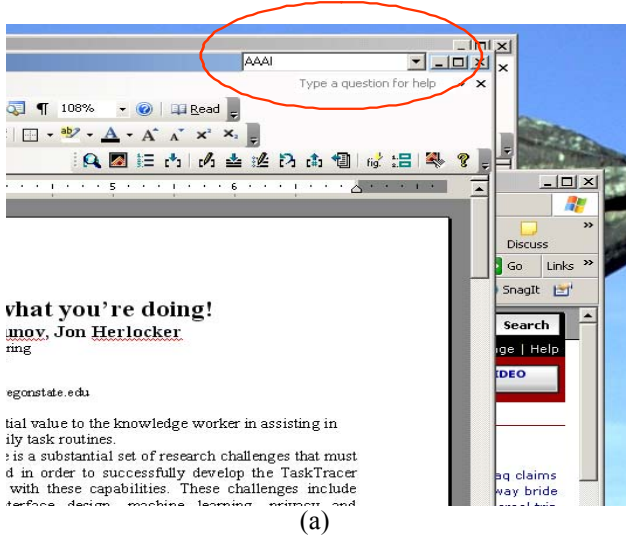


Fig. 1 – TaskPredictor predicts tasks on windows focus switches

The probabilistic framework we are employing in TaskPredictor can be outlined as follows. Let observation  $O = (o_{t-k}, \dots, o_{t-1}, o_t)$  be an ordered set of observations from time  $t-k$  to  $t$ , where  $k$  is 0 if we ignore the temporal relationship and only consider the current observation. Our goal is to get a probabilistic distribution about the current task given  $O$ :  $P(\text{Task}_t = \text{task}_i | o_{t-k}, \dots, o_{t-1}, o_t)$ .

Feature construction occurs as follows. A Window Document Segment (WDS) consists of the time period in which a window has the focus and this window is looking at a single document. It is assumed that the user is on a single task in the same WDS and a prediction is only necessary when the WDS changes. We make a prediction when navigation events occur in Internet Explorer, window focus switches, when a new application is started, or a resource is opened or saved. The source for the features comes from window titles, file pathnames, website URLs, and document content. Each source is segmented into a set of “words”, where each word corresponds to a binary variable  $w_i$  in the feature vector. We utilize a stopwords list to eliminate irrelevant features. We then use a Naïve Bayesian classifier to learn  $P(w_i | \text{task}_i)$  and  $P(\text{task}_i)$  and make predictions by using Bayes rules to calculate  $P(\text{task}_i | w)$ .

We have evaluated this approach by testing on a dataset from a team member. This data set, collected over a period of three months, contained 81 different tasks, 11455 WDS and 1239 features. If we predict on every WDS ( $\theta = 0$ , where  $\theta$  is the normalized probability of a task that is computed from the Naive Bayes model), using unprocessed data, we achieve an accuracy of 25%. Once “meaningless” events – events that happen in all tasks (e.g. open/save dialogs, blank web pages) or events not related to any tasks (e.g. a file used by an application all the time) – are removed, the accuracy is further increased to 60% (see Fig. 2).

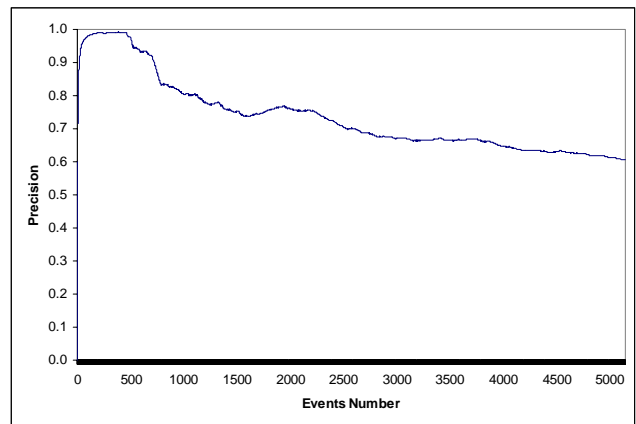


Fig. 2 – TaskPredictor precision results after removing “meaningless” data

If we abstain from making a prediction when we are not confident (when we are below a threshold  $\theta$  of the normalized probability of a task) we further increase the level of correct predictions to 85% ( $\theta = 0.9$ ) (see Fig. 3).

However, whether or not 50% coverage is good enough to the user needs to be investigated, but not every WDS is a task switch so an abstention might be the right answer.

We have compared Logistic Regression and Dynamic Bayesian Networks (DBNs) to the Naïve Bayes (NB) approach. If we always make a prediction (that is, the threshold is 0), then Logistic Regression significantly outperforms its generative analog the Naïve Bayes classifier. However, with other thresholds their performance is on a par and, since the training of Logistic Regression is more expensive, the Naïve Bayes approach is preferable. DBNs can capture the temporal relationship, since DBNs use all available observations. Common sense tells us that it therefore should be more suitable for TaskTracer. However, in our situation, it does not result in increased accuracy over Naïve Bayes. One possible reason for this is that in most sequential problems, people make predictions based on the whole information (past, current and future observations), but in TaskTracer we can only use the information up to a certain point (past and current observations). Again, according to Occam's Razor, since DBNs and NB have similar accuracy, and the learning and inference of DBN are much more expensive than NB, we prefer NB.

Finally, by applying feature selection by using information gain, we appear to have pushed the accuracy to 95%.

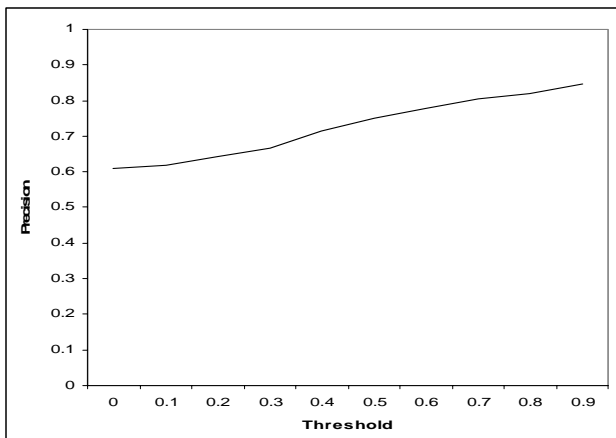


Fig. 3 – TaskPredictor precision results using thresholds for prediction abstentions

Once we know the current task, we leverage that information to support user activity. We have implemented a FolderPredictor to reduce the user cost of accessing resources given a certain task. Knowledge workers often have different folders for each task. In our approach we sort the folders based on the frequency in which user has opened/saved files before and use exponential decay to prioritize most recently accessed directories. For the recency weighting in FolderPredictor a parameter named DiscountFactor is used, which is a real number between 0 and 1. Each time the predictions are updated, the old predictions are multiplied by this DiscountFactor. We

tuned this parameter by experiment, and found that 0.7 works best on the test data.

We employ these predictions by changing both the initial folder and the places bar of the Windows Open and Save As dialog box, as shown in Fig. 4. At the moment the usefulness of the prediction is evaluated by the cost to the user in reaching the desired file i.e. the distance between predicted folders and the user's destination folder. We find that, on average, the desired folder is roughly one click away from the set of folders returned by FolderPredictor.

On the surface, the TaskTracer approach is similar to MailCat or SwiftFile (Segal and Kephart 1999, 2000) which predicts folders in which to store emails. MailCat employs TF-IDF by training on emails filed previously under folders by the user. However, our approach is different in the following aspects:

- A prediction is made for any Microsoft Office resource, not just emails.
- Appropriate folders are tied to user-defined task names. We train on resources being opened and saved under particular tasks (a resource can be associated with more than one task) instead of documents being explicitly filed under folders by the users.
- Our approach hooks directly into the native Windows environment; it is transparent to the user and therefore carries no overhead cost for the user to learn additional interactions.

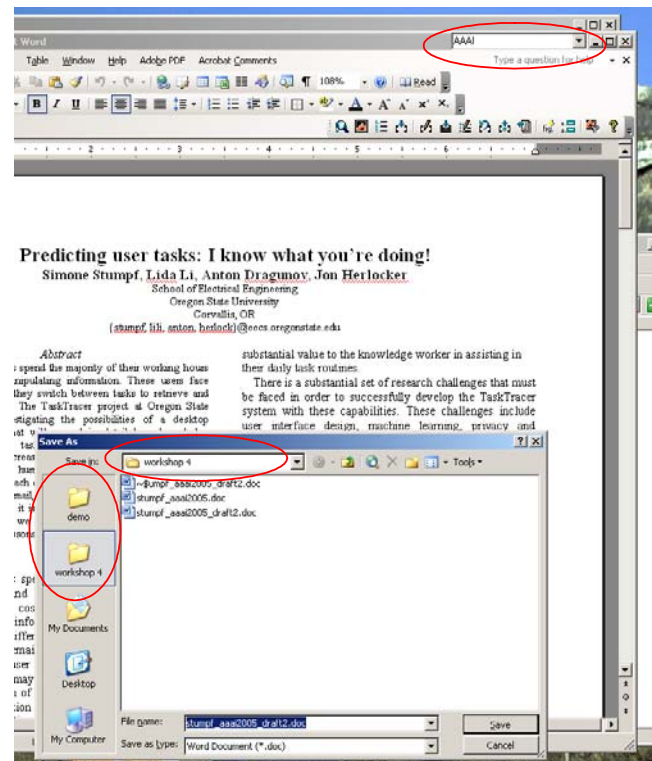


Fig. 4 – FolderPredictor integrated into the Open dialog box

## Discussion, Future Work and Conclusion

TaskPredictor, FolderPredictor and the associated TaskTracer system have been used by our research team for more than six months. FolderPredictor in particular appears to be the most cost-saving component in this environment. We are in the process of conducting usability evaluations “in the wild” to gain feedback from real knowledge workers and hope to publish our findings in due course.

While we are addressing the machine learning component we realize that there are still some issues, particularly on making machine learning more comprehensible by humans. Firstly, we need to address the user’s perceived loss of control when predictions are made. This suggests that we need to investigate the appropriateness of different prediction strategies related to user cost. For example, one such strategy would be to only offer these predictions as suggestions to the user. However, additional cost is then placed on the user in choosing the appropriate prediction. Another approach that we considered is to make predictions only when the accuracy is high enough. The approach that we currently take makes predictions when the predictions are of high confidence and abstains from predictions when the confidence is low. It appears that good results can be achieved with this approach but further testing will confirm the acceptability by real users.

Secondly, we are considering how we can make the reasons for why a prediction was made more comprehensible. Currently, we are projecting a simple system model to the user that explains why a certain prediction is made. It is based on what resources were accessed in episodes and when manual task switches happened. The user can relate to the fact that they “forgot” to switch the task and therefore associated certain activities with the wrong task. We can exploit a mental model of how task switching occurs by providing a timeline or simple list of events associated with tasks to provide an explanation to the user of why predictions were made and what they are based on. Instead of using a rule-based explanation of predictions we present the grounds of a prediction informally.

Lastly, we are looking at how feedback from the user can be utilized by the learning components. We have to realize that users are “lazy” positive example givers; we have addressed this by making users give us implicit examples as they complete their tasks. We are already using online learning to enable corrections to affect the how predictions are made. More interestingly, users are often not 100% sure themselves or may provide different answers in different contexts. Users are often able to tell the system what it is not, but not what it is. For example, consider a suggested list of tasks where the user indicates that it is none of these but does not specify the correct one. How this should be used by learning components is still an unresolved problem. Furthermore, we are looking at getting the user to provide examples: it seems that the more useful things TaskTracer does, the more motivation the

user will have for telling us their tasks. For example, if FolderPredictor is used all the time, it gives implicit feedback by displaying the wrong prediction if the user forgets to change the current task. So maybe the way to motivate the user is just to keep providing useful features and rewarding them for their costs. Of course, there is the danger that the user may have no idea at first that such rewards are in store. It may look instead like a bunch of costs for no particular reason. If that happens, the user could ignore the system's "faulty feedback" (or turn it off).

To reduce user costs we are looking at using machine learning approaches. In this paper we have discussed TaskPredictor and FolderPredictor, that can reduce user cost. What is important is that both components use user input to update their model. Users can easily correct the task or folder predictions and TaskPredictor and FolderPredictor use this information for on-line learning. We have achieved some initial encouraging results and are currently working on refining our approaches. This will include extensive user testing and development of our machine learning algorithms.

## Acknowledgements

This project was supported in part by the National Science Foundation under grant IIS-0133994 and by the Defense Advance Research Projects Agency under grant HR0011-04-1-0005.

## References

- Bannon, L., Cypher, A., Greenspan, S. and Monty, M. 1983. Evaluation and Analysis of Users' Activity Organization. *Proceedings of the ACM CHI 83 Human Factors in Computing Systems Conference*, Boston, Massachusetts, ACM Press.
- Bauer, M. 1999. Generation of Alternative Decompositions for Plan Libraries. *IJCAI'99 Workshop on Learning about Users*.
- Bellotti, V., Ducheneaut, N., Howard, M. and Smith, I. 2003. Taking Email to Task: The Design and Evaluation of a Task Management Centered Email Tool. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, ACM Press.
- Canny, J. 2004. Gap: A Factor Model for Discrete Data. *Proceedings of SIGIR*, ACM Press.
- Card, S. and Henderson, A. 1987. A Multiple, Virtual-Workspace Interface to Support User Task Switching. *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, Toronto, Ontario, Canada, ACM Press.

Davison, B. D. and Hirsh, H. 1998. Predicting Sequences of User Actions. *AAAI-98/ICML'98 Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, Madison, WI.

Dourish, P., Edwards, K., LaMarca, A. and Salisbury, M. 1999. Presto: An Experimental Architecture for Fluid Interactive Document Spaces. *ACM Transactions on Computer-Human Interaction (TOCHI)* 6(2): 133-161.

Dragunov, A., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L. and Herlocker, J. L. 2004. Tasktracer: A Desktop Environment to Support Multi-Tasking Knowledge Workers. *International Conference on Intelligent User Interfaces*, San Diego.

Fenstermacher, K. D. and Ginsburg, M. 2002. A Lightweight Framework for Cross-Application User Monitoring. *IEEE Computer* 35(3): 51-59.

Freeman, E. and Gelernter, D. 1996. Lifestreams: A Storage Model for Personal Data. *ACM SIGMOD Record* 25(1): 80-86.

Kaptelinin, V. 2003. Umea: Translating Interaction Histories into Project Contexts. *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, ACM Press.

Ourston, D. and Mooney, R. J. 1990. Changing the Rules: A Comprehensive Approach to Theory Refinement. *8th National Conference on Artificial Intelligence*, Boston, MA.

Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Ridsen, K., Thiel, D. and Gorokhovskiy, V. 2000. The Task Gallery: A 3d Window Manager. *Proceedings of the SIGCHI conference on Human factors in computing systems*, The Hague, The Netherlands, ACM Press.

Segal, R. and Kephart, J. 1999. Mailcat: An Intelligent Assistant for Organizing E-Mail. *Proceedings of the Third International Conference on Autonomous Agents*.

Segal, R. and Kephart, J. 2000. Incremental Learning in Swiftfile. *Proceedings of the Seventh International Conference on Machine Learning*.