Understanding Desktop Users: Practical Applications of Machine Learning For Predicting User Action

Ian Cohen icohen@seas.upenn.edu
James Walker jswalker@seas.upenn.edu
Faculty Advisor: Lyle Ungar ungar@cis.upenn.edu

April 18, 2008

Abstract

In this project we attempted to employ standard machine learning methods on the linux desktop with the goal of answering the question, given the user’s past behavior, What will the user do next? The major components of the project were to design a desktop event correspondance scheme, build a plugin architecture for registering these events, and implement a learner which predicts new actions. It was intended as a sister project to GNOME Dd\footnote{2007-’08 CIS 400/401 Project by Douglas Colkitt and David Siegel. In short, a sophisticated action launcher for the linux desktop. \url{http://do.davebsd.com/}} which can use our output as a component in their ranking scheme, and provide us with on-line feedback. The system has been named predix.
Contents

1 Example Use Cases 3

2 Related Work 3
   2.1 TaskTracer ................................................. 3
   2.2 Probabilistic Online Action Prediction .................. 4
   2.3 Action Prediction for Intelligent Homes ................. 4

3 Work to-date 4
   3.1 Features and Components ............................... 4
   3.2 Omissions ................................................. 5

4 Technical Approach 6
   4.1 Overview .................................................. 6
   4.2 D-Bus ..................................................... 7
   4.3 Data Sources .............................................. 7
   4.4 Learning .................................................... 8
      4.4.1 Features for Learning ................................. 8
      4.4.2 Algorithms ........................................... 8
      4.4.3 Performance .......................................... 9
   4.5 Third Party Integration ................................... 10
   4.6 Challenges and Difficulties .............................. 11

5 Conclusion 11

References 12
1 Example Use Cases

- Carli is a GNOME Do user. She sends mail to her best friend Sam more often than she sends mail to her other friends Sally and Sandra. With predix running, Carli no longer has to filter her contacts in Do by typing ‘S-a-m’. Instead, she types ‘S’ and Sam is the top contact presented to her by Do.

- John begins his mornings with a visit to Traffic.com to check on local road conditions. He is uninterested in traffic news after returning from work so he does not want to make Traffic.com his homepage. He downloads a Firefox extension which populates the browser’s url list with relevant urls from predix and gives him the option of automatically loading the top ranking page. predix suggests Traffic.com as the most likely website in the morning until it is visited. At this point its rank decreases, preventing its accessibility from becoming an inconvenience.

- Jane loves to listen to music as she chats with friends online. We learn that if Jane starts an instant message program, her next action is likely to be launching her music player. predix puts Jane a keystroke away from starting her music when she most feels like listening to it.

2 Related Work

2.1 TaskTracer

TaskTracer\(^2\) tracks user actions to build predictive models of user tasks. The TaskPredictor component associates user actions with user-inputted task labels. It learns to predict the label of the current task and identify task switches. The FolderPredictor component returns the most relevant folders for open/save actions, given the current task.

TaskTracer is built on the assumption that “knowledge workers organize their work into discrete and describable units, such as projects, tasks or to-do items”\(^1\). predix does not rely on this assumption and we do not aim to classify user actions into discrete categories. Furthermore, we do not place the ontological burden on the user. Instead, we recognize that user context is continuously changing and requires a complex fine-grained analysis. Also, as

\(^2\)http://eecs.oregonstate.edu/TaskTracer/
explained above in the Use Cases (Section 1), we do a lot more than suggest relevant folders.

2.2 Probabilistic Online Action Prediction

Davison and Hirsh[2] use online learning to predict a user’s next UNIX command, given the user’s command history. predix is a multi-dimensional expansion of this work. The feature-set used by Davison and Hirsh consists of command texts and command order. We harvest a much richer hierarchy of information, as discussed in Section 4.

2.3 Action Prediction for Intelligent Homes

We have discovered a body of work on action prediction in intelligent homes. While homes are not our domain of interest, we believe we can learn from work that models and clusters actions in a feature-rich environment[3].

3 Work to-date

At this point, predix has most of the functionality we set out to implement. predix collects information on user’s actions and other events in the system, according to an extensible plugin system, can perform many learning algorithms on the data, and can make its results accessible to subscribing applications through another plugin system.

3.1 Features and Components

predix contains the following components:

- D-Bus proxies
- Logger
- Context collector
- Context collector plugin system:
  - System time
  - System uptime
• Current user
• Current focused window

• Modules for integration with:
  – GNOME Panel
  – GNOME Do
  – Mozilla Firefox

• Extensible feature generator

• Prediction Algorithms:
  – Last action
  – Most frequent action
  – Incremental Probabilistic Action Modeling
  – Naive Bayes
  – K-Nearest Neighbors

3.2 Omissions

We did not complete all the work we set out to do. We did not run a “study” of user behavior. Distributing all components necessary for learning proved to be beyond the scope of the project. The high number of system customizations required for data collection made it infeasible to create a self-installable package and distribute it to volunteers, especially considering possible privacy issues and the need for our processes to be running at all times. We realized that the amount of data necessary to have proper learning would be orders of magnitude greater than the amount we could collect in such a user study. Instead, we focused on building the infrastructure necessary to learn from data which could later be collected on a professional scale. We tested this system with simulated user data in accordance with our use cases.

We did not construct an ontology of the nature we had planned. Given the small domain of actions on the desktop, our proposed ontology proved unnecessary. Instead use a light-weight tagging system. Determining the type of any event that we have collected is trivial based on simple properties
of the data. Furthermore, if an ontology were required for future learning algorithms, it too would be trivial to implement given our plugin-based structure.

4 Technical Approach

4.1 Overview: Analysis of the Use Cases

• No third-party plugins are necessary for predix to recognize that Carli is more likely to send mail to Sam than to Sally or Sandra. Do publishes signals on D-Bus with unique identifiers of the command and items involved in each action. predix listens to these signals and stores them in an event history database. It then analyzes the database to discover that the mail action is performed upon Sam’s contact entry with greater frequency and consequently increases the rank of Sam’s contact for the mail action. This increase in rank is recognized by Do next time Do populates Carli’s contact items since Do requests relevant ranking information from predix for each action.

• John’s use case demonstrates that predix can empower applications beyond Do. In this case, we need a plugin to capture browsing history by receiving url visit events. Firefox requires an extension to request url ranking information from predix and populate the url list. This use case also demonstrates our ability to identify chronological usage patterns. Time of day is one feature that is be derived from the context collected by the context logger. Other features include focused window, idle time, and of course n-grams of past user actions.

• Jane’s use case has a similar explanation to John’s use case. The difference between the use cases lies in the type of association being performed. John’s actions were associated with a feature of the environment while Jane’s actions are associated with other actions. This illustrates the importance of the downstream feature set as explained below.
4.2 D-Bus

D-Bus is a service for inter-process communication on Unix-like systems. It provides two style of communication, \textit{signals} and \textit{methods}. Signals are emitted by applications and can be subscribed to by others. Methods are explicit queries made by one application to another, which can optionally return some data.

D-Bus is used in several ways in \textit{predix}. The event logger listens to D-Bus signals as its primary source of events. The context logger makes D-Bus method calls to collect information from various parts of the system. We have a proxy object on the D-Bus which acts as a service for other applications that want to use our learning. It is a way they can make requests for relevant objects or rankings.

4.3 Data Sources: Logging events and collecting context

\textit{predix} distinguishes two kinds of data: events and context. An \textit{event} is a change in the environment which is pushed to our system. \textit{Context} collectively describes those data which our system must pull from other sources. We poll for context at regular intervals, and also upon certain events, which we will call \textit{target events}. We have decided, however, to treat all events as target events; that is, after we are notified of any event, we collect the context.

An example of an event is when the user’s status changes from idle to active, after waking up the computer or returning from the screensaver (published onto D-Bus anyway), or when a user performs an action like sending mail or opening a file (requires third party adaptations). Examples of context data are current time, information about the current focused window, and what urls are open in the web browser (obtained through plugins).

An event logger records all information about any events pushed to us, and prompts the context logger to fetch all the desired context information to be recorded with the event.

The event logger receives signals from the D-Bus session bus. The event logger can be extended by identifying which signals to receive.
4.4 Learning

4.4.1 Features for Learning

The data on which we run machine learning algorithms goes through several stages. Context data (as described above) is thought of as “upstream” features, such as time, CPU usage, and current focused window. From the context and event logs we will derive “downstream” features, which correspond with what is traditionally called a feature in machine learning. This includes features such as the amount of time since the user came back from idle (calculated from the no-long-idle event), the uptime at a given event (calculated from the initial context), and $n$-grams of actions, as mentioned earlier.

We have built a framework for adding new features in a simple way, given objects encapsulating the event and context data. We automatically generate meta-features such as $n$-grams of other features in the list. It would be simple to extend this to allow third parties to add features, but that is of questionable utility. Perhaps savvy users might like to customize the learning process.

4.4.2 Algorithms

We used the Orange [4] machine learning framework to implement Naive Bayes, k-Nearest Neighbors, and Most Frequent Action learning algorithms. Orange provides model validation techniques (e.g. cross-validation, random sampling, leave-one-out, etc.) which allow us conveniently compare the results of various learners. We did not use Orange for our online learner IPAM since IPAM does not use all of the features we collect. IPAM is an adjustable algorithm which makes a Markov assumption that each action depends only on the previous action. It stores a table of transitions between actions and updates that table based on observations of the next command. The parameter $\alpha$ adjusts the exponential rate of decay of the update function. An $\alpha$ of 0 means that the algorithm predicts what it saw most recently for the action transition, whereas an $\alpha$ of 1 means that the algorithm keeps a uniform distribution of transition probabilities.
4.4.3 Performance

On the limited data we collected, k-Nearest Neighbors outperformed the baseline of Most Frequent Action. We have not collected enough data to properly test our algorithms but these preliminary tests serve as a proof of concept.

<table>
<thead>
<tr>
<th>original</th>
<th>MFA</th>
<th>k-NN (k=5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>7.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>3.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>7.0</td>
</tr>
<tr>
<td>9.0</td>
<td>12.0</td>
<td>7.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1.0</td>
<td>12.0</td>
<td>3.0</td>
</tr>
<tr>
<td>6.0</td>
<td>12.0</td>
<td>7.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>2.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>7.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>14.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td>4.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>12.0</td>
<td>2.0</td>
</tr>
<tr>
<td>15.0</td>
<td>12.0</td>
<td>14.0</td>
</tr>
<tr>
<td>7.0</td>
<td>12.0</td>
<td>1.0</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td>14.0</td>
</tr>
</tbody>
</table>
### 4.5 Third Party Integration

Many of the features of predix rely on data coming from outside sources, or the ability of other applications to respond to suggestions. Several kinds of plugins can be used to solve these problems.

The context logger is easily extensible with a lightweight plugin-like system whereby a third party can identify a new function whose return value will be stored when context is collected. With respect to the machine learning component, these can be thought of as upstream features. Any methods labeled `@context_method` which are in the Addins directory will be used to collect context. For example, our addin which lets us know the current window is as follows:

```python
import Xlib.display
from Learner.Addins import context_method
display = Xlib.display.Display()

@context_method
def focused_window_name():
    window = display.get_input_focus().focus
    if isinstance(window, int):
        return window
    else:
        return window.get_wm_name()
```

Additionally, applications can extend their own functionality in two ways: put notifications on D-Bus, and ask for suggestions. We have developed scripts which cause the GNOME Panel to send D-Bus notification whenever

---

<table>
<thead>
<tr>
<th>Learner</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFA</td>
<td>41.423</td>
</tr>
<tr>
<td>k-NN (k=5)</td>
<td>11.887</td>
</tr>
</tbody>
</table>

---

1.0  12.0  10.0  0.0  12.0  14.0  11.0  12.0  11.0  14.0  12.0  8.0  6.0  12.0  6.0  12.0  10.0
a panel button is used, which is ideal for tracking user actions outside of GNOME Do. We have also developed a Mozilla Firefox extension which puts URLs on the D-Bus as they are visited, and allows the user to have predix suggest the next URL to visit.

We maintained two-way communication with GNOME Do. We can offer and display a default suggestion for a common set of actions and items.

4.6 Challenges and Difficulties

The largest challenge has been learning to use the APIs we need. Documentation is sparse and it is often a game of guess and test. For example, to receive D-Bus signals one has to set up a certain kind of busy loop (as noted in the documentation). What was not noted anywhere was that running this loop in its own thread will break all other threads. The fix (to configure the loop for threading), was found in some mailing list archives. In order to get GNOME Do to send the appropriate signals, we needed to use dbus-sharp, the C# binding for D-Bus, which lacks any documentation whatsoever. It was only after contacting one of the dbus-sharp developers using IRC that we were able to proceed.

Collecting data proved much more difficult than expected, as mentioned above.

Another difficulty was maintaining compatibility with GNOME Do as it underwent large architecture changes over the course of the year, but understanding their motivations helped us to update our own project.

5 Conclusion

As documented in Work to-Date (Section 3), we have met most of the goals we set out for ourselves. predix is a fully functioning machine learning desktop prediction system, fully extensible by third parties, and ready for professional testing on large data sets with fancier plugins. We are very excited at the prospect of determining which algorithms perform better on real data. It will truly be interesting if plugins are designed which can, for example, notify when a panel menu is opened but no buttons are selected. predix will immediately be able to harness that new information.
References


This is the author’s version of a paper published in the proceedings of AAAI-05, the premier showcase of AI science and technology. The paper discusses TaskTracer, A Task-Based Interruption Recovery & Knowledge Management Tool. The project is part of the Management of Knowledge-Intensive Dynamic Systems initiative at Oregon State University. Dr. Simone Stumpf discusses the task-oriented intelligent system that tracks user’s interactions with applications, organizes the user’s information naturally according to tasks, and intelligently leverages the collected data to make desktop applications more task-aware. TaskTracer and its sister project FolderPredictor ask users to label their current work environments as discreet tasks and consequently deviate from our unobtrusive learning methods. Nevertheless this paper was relevant in its thoughtful considerations of the conveniences and pitfalls of an action prediction system.


In this paper, Professor Brian D. Davison of Leigh University and Professor Haym Hirsh, Director of the Division of Information and Intelligent Systems at the National Science Foundation propose characteristics of an idealized algorithm that would allow a user interface to adapt to an individual’s pattern of use. They present their own ideal online learning algorithm, Incremental Probably Action Modeling Algorithm and show its predictive accuracy on a large dataset of UNIX commands. We adapt and implement IPAM for online learning of our data.

Dr. Diane Cook, Huie-Rogers Chair Professor at Washington State University and professor Sira Panduranga Rao of University of Texas at Arlington describe a simple Markov model for predicting the next action of inhabitants of intelligent homes. We used this paper to learn from an study of action prediction in a feature-rich environment. They discuss the shortcomings of traditional machine learning techniques when applied to sequential prediction problems using historic information.