# Toolkit Support for Developing and Deploying Sensor-Based Statistical Models of Human Situations

**James Fogarty**
Computer Science & Engineering
University of Washington
Seattle, WA 98195
jfogarty@cs.washington.edu

**Scott E. Hudson**
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
scott.hudson@cs.cmu.edu

## ABSTRACT
Sensor-based statistical models promise to support a variety of advances in human-computer interaction, but building applications that use them is currently difficult and potential advances go unexplored. We present Subtle, a toolkit that removes some of the obstacles to developing and deploying applications using sensor-based statistical models of human situations. Subtle provides an appropriate and extensible sensing library, continuous learning of personalized models, fully-automated high-level feature generation, and support for using learned models in deployed applications. By removing obstacles to developing and deploying sensor-based statistical models, Subtle makes it easier to explore the design space surrounding sensor-based statistical models of human situations. Subtle thus helps to move the focus of human-computer interaction research onto applications and datasets, instead of the difficulties of developing and deploying sensor-based statistical models.

### Author Keywords
Toolkits, Subtle, sensor-based statistical models, machine learning, context-aware computing.

### ACM Classification Keywords
H5.2. Information interfaces and presentation: User Interfaces; H1.2. Models and Principles: User/Machine Systems.

## INTRODUCTION AND MOTIVATION
Context-aware computing promises applications that sense an environment, model situations, and act appropriately [4]. For example, consider that traditional email clients with no understanding of context often use an audio notification to announce new email. Appropriate and useful when alone in a private office, this can be disruptive and awkward during a meeting with colleagues. While current applications are generally unaware of such contexts or the impact of their actions, context-aware applications could infer whether such a notification is currently appropriate.

Many contextual cues are unambiguous (a sensor is either activated or it is not), but the ultimate decision about how an application should behave is often inherently ambiguous. No single sensor directly maps to "an audio notification is currently inappropriate" and it is unreasonable to expect users to specify what actions should be taken in every conceivable context. *Sensor-based statistical models* are one solution to this problem. Developed and deployed, a sensor-based statistical model can give an application insight into the situation surrounding its usage, providing guidance on what actions are likely to be appropriate.

Successfully deploying sensor-based statistical models is currently difficult. The problem is typically *not* that models are incapable of learning about human situations, as the machine learning community has developed powerful methods for learning from labeled datasets [30, 34]. However, the use of machine learning generally requires significant specialized knowledge, particularly with regard to bridging the gap between sensed context and high-level features appropriate for use in machine learning algorithms. Furthermore, existing tools are generally designed for use with previously collected data, neglecting the requirements of model deployment. The human-computer interaction research community has the insights needed to develop compelling applications of sensor-based statistical models, but the barriers to working with them are currently high. Potential advances therefore go unexplored.

This paper presents Subtle, a toolkit for *Sensing User Behavior To Learn about the Environment*. By reducing the barriers to model development and deployment, Subtle makes it easier to explore the design space surrounding sensor-based statistical models of human situations. For example, we have used Subtle to build an application that learns to automatically toggle whether a laptop's audio is muted. Each time a person manually toggles their audio, the application provides Subtle with a label indicating the preferred setting in the current context. Subtle then uses the provided labels to learn a sensor-based statistical model (considering such factors as a person's location and what applications they are using). Built with Subtle, this application uses just 16 lines of code to collect appropriate sensor data, learn a model, and obtain live estimates from that model. This result, the ability to build an application

that can monitor, learn from, and respond to sensed context with just a handful of code, represents an important contribution. By reducing the barriers to using sensor-based statistical models of human situations, Subtle makes it easier for the human-computer interaction community to explore the potential of such models in applications.

The next section details three specific difficulties in using sensor-based statistical models and provides an overview of how Subtle addresses each. We then review related work, including a differentiation between Subtle and previous toolkits for context-awareness or machine learning. We next present Subtle's implementation, showing how we support the development and deployment of sensor-based statistical models. We then discuss the validation of Subtle, considering its performance on two previously published datasets, giving two examples showing that Subtle enables application development with very little code, and reporting on other researchers using Subtle in their own work. Finally, we present a short conclusion.

## SENSOR-BASED STATISTICAL MODELS

Subtle supports supervised learning, wherein *labels* are collected when a value is known for the concept being modeled (as when a user indicates a preference). A model is learned by extracting statistical relationships between the values of labels and *features* of the sensed context at the time of each label. Based on our experience developing and deploying sensor-based statistical models of human situations [9, 10, 11, 12, 22] and the experiences of others [2, 3, 4, 6, 16, 17, 18, 19, 20, 24], at least three problems must be addressed in a tool to support the development and deployment of sensor-based statistical models.

*Learning statistical models requires sensors that can provide relevant context.* If the available sensors are not related to a concept being learned, no amount of processing can extract a meaningful relationship. This can ultimately be addressed only by an application designer, as it is the application designer who chooses what concept will be learned from what sensors. Subtle takes a two-pronged approach to this problem. First, Subtle provides a library of sensors that are useful in many applications. Targeted at a typical laptop computer, this library currently includes analyses of the desktop event stream, analyses of ambient audio, and WiFi location sensing. Second, Subtle's data collection mechanisms are designed to support extensibility, making it easy to add new sensors.

*It is hard for a static model to account for individual differences and unexpected situations.* There are often important differences in how people want an application to behave. For example, our prior work found that researchers who program are typically less interruptible when actively working on their computer, while first-level managers are typically more interruptible in the same situation [9]. If a model cannot adjust to such differences, its utility will suffer. Subtle addresses this by supporting continuous and fully-automated learning of personalized models on an end-user's computer. Because Subtle can continuously learn models from labels provided by an individual end-user, applications can adapt to personal preferences.


**Figure 1. Whistle addresses the problem that people often forget to mute or unmute laptop computers. It collects a label each time a person manually toggles the mute flag, learning a relationship between context and a person's desired setting.**

*It is hard to determine what aspects of available context will be most useful in a model.* Sensed context is rarely appropriate for direct use in a machine learning algorithm, and must instead be processed to extract high-level features appropriate for learning. Non-experts can be very unsure about what types of features are appropriate, and even experts are unlikely to manually craft the best possible features [29]. While this problem is important in many applications of machine learning, Subtle's need to learn models in a background process on an end-user's machine makes this problem especially important, as no expert is available to intervene in the automated model learning process. Subtle is therefore based on a type-based approach to iterative feature generation, applying operators to automatically generate a large number of potential features for consideration in a feature selection algorithm. This approach promotes extensions of Subtle's sensing and learning mechanisms and removes the need for an application designer to acquire significant specialized knowledge of machine learning techniques.

## RELATED WORK

Several groups in the human-computer interaction research community have examined sensor-based statistical models of human situations, with interruptibility and availability being common topics of interest [1, 2, 15, 16, 17, 18, 19, 20, 23, 24, 31]. The transition from studying models of human situations to deploying systems based on such models introduces additional requirements and difficulties. Horvitz *et al.* have deployed Priorities, Coordinate, BusyBody, and BestCom, all of which use sensor-based statistical models to reason about interruptibility and availability [16, 17, 18, 19, 20]. Begole *et al.* deployed their temporal models of availability in Awarenex [2]. We deployed and evaluated a context-aware instant messaging client that includes an availability model [12]. Because it is currently difficult to deploy sensor-based statistical models of human situations, evaluations of such deployments are often focused on technical innovations or on analyses of model reliability in deployed applications. This neglects

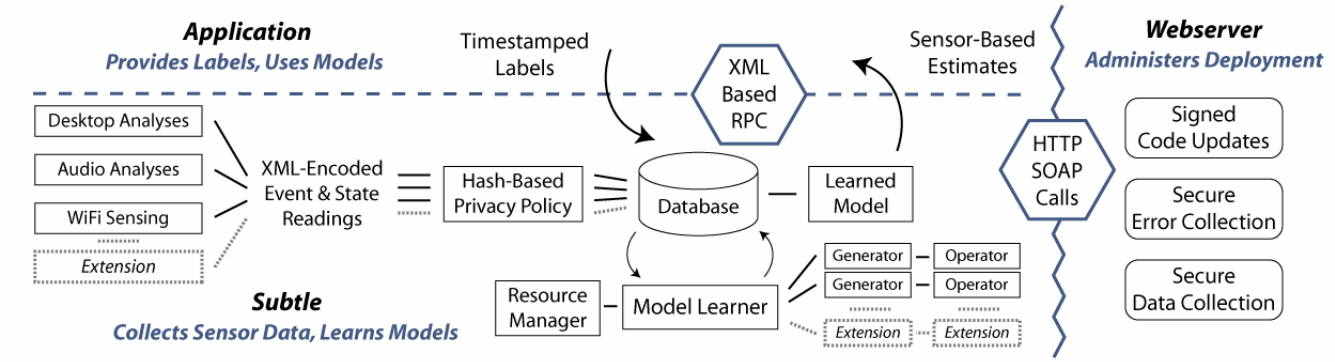


**Figure 2. Overview of Subtle's runtime architecture.**

many compelling human-computer interaction research questions, such as how a model can most improve an interaction, how to design compelling applications despite the inevitable errors made by statistical models, and what impact models have on how people interact with computers. By making sensor-based statistical models available to a larger audience, Subtle encourages research into these types of questions, which are fundamentally different from the questions typically explored by people whose primary interest lies in machine learning.

Two important categories of related toolkit work exist. The first is work on toolkits to support context-awareness, the most relevant example being Dey *et al.*'s Context Toolkit, which supports the distributed collection, storage, and retrieval of context [4]. Designed for non-ambiguous context, the Context Toolkit has been enhanced to support the mediation of ambiguity [3]. With mediation, an application prompts a user to resolve ambiguity. For example, Dey *et al.* present the Communicator application, which uses context to choose among language models for a word completion widget. If it is ambiguous which model should be used, a prompt is displayed for the user to choose a model. While mediation can support many applications, it is clearly inappropriate for many others. In the case of audio notifications, mediation could result in prompts asking whether it is appropriate to deliver an audio notification (an approach that is likely more disruptive than the original notification). More importantly, the Context Toolkit does not learn from mediation, so people may often be asked to resolve identical or very similar situations.

Existing machine learning toolkits represent a second important category of related work, with Weka being one of the most widely known and used [34]. Designed by and for machine learning researchers, Weka and similar toolkits provide a variety of feature selection and machine learning algorithms, but require significant specialized knowledge of machine learning before they can be effectively used. Such tools are intentionally agnostic to how input features were computed, typically assuming that a researcher is analyzing files that contain data previously collected and processed to extract meaningful features. Because these tools assume appropriate high-level features as input, they do not provide support for helping to construct such features.

We note that Subtle's research contributions go beyond an aggregation of these two types of existing toolkits. Subtle necessarily overlaps these tools, including both mechanisms to collect, store, and retrieve sensed context as well as a set of feature selection and machine learning algorithms. But adding the capabilities of Weka to the Context Toolkit would not provide a toolkit equivalent to Subtle, as neither existing tool provides support for constructing appropriate high-level machine learning features from low-level sensed context. Subtle is unique in providing a type-based approach to fully-automated feature generation, and its abstractions are unique in allowing application developers to develop and deploy sensor-based statistical models of human situations without first acquiring extensive specialized knowledge of machine learning.

The prior work most relevant to Subtle's automated feature generation is an iterative approach to automated feature construction presented by Markovitch and Rosenstein [29]. Their approach is based in a formal grammar specifying legal manipulations of a base feature set. Because their approach is intended for machine learning researchers, the manual creation of an appropriate grammar requires both significant specialized knowledge of machine learning and of the specific learning problem to be solved. In contrast, Subtle's automated feature generation is built upon a set of abstractions that both enable non-expert use and are also appropriate for learning a variety of concepts based on the types of sensed context considered by Subtle.

## SUBTLE ARCHITECTURE

Figure 2 presents an overview of Subtle's architecture. We will discuss each component of this architecture in detail as this paper progresses, but this section provides a high-level overview as a basis for future sections. Implemented in Java with approximately 20,000 method lines of code, Subtle is a service that, once started, runs in a separate process from an application. Applications interact with Subtle using XML-encoded remote procedure calls (a provided wrapper makes this communication transparent to Java programs). When Subtle is started, it spawns a variety of sensors, each of which begins to generate XML-encoded *readings*. Each reading is either an *event* or a *state*, where events occur in an instant and states retain their value until either a new value arrives or a timeout occurs. Collected readings are

subjected to a *privacy policy* before being stored in a database. An application is then responsible for providing timestamped labels, typically obtained when an end-user provides an indication of the desired behavior of an application. At regular intervals, the *model learner* examines sensor logs and the collected labels to learn a new statistical model. The learner uses *generators* to iteratively examine potential features and automatically creates new potential features by applying *operators* to transform existing potential features. Once a model has been learned, it can be evaluated against the database to provide live *estimates*. Finally, a set of web services support the field deployment and data collection from a Subtle application.

## EXTENSIBLE SENSING LIBRARY

Subtle takes a two-pronged approach to addressing the problem that *learning statistical models requires sensors that can provide relevant context.* The first part of our approach is to provide a library of generally useful sensors. In building this library, our focus has been on sensors that can be deployed in software on a typical laptop computer. While custom hardware is clearly important to some applications, focusing on software sensing maximizes Subtle's relevance to everyday use. Sensing is implemented using standard Microsoft Windows libraries, so it works with appropriate applications and hardware on a dominant platform. Subtle currently provides desktop event stream analyses, ambient audio analyses, and WiFi location sensing. This library will grow as additional sensors are developed for specific applications, and adding new sensors to this library will make them available to every application built with Subtle. Our provided sensing library addresses the most common case, and Subtle's extensibility addresses the use of sensing infrastructure or custom hardware.

### Provided Sensor Library

Our desktop event stream sensing library monitors several indications of how a person is using their computer. For each top-level window, Subtle logs the window title, type, and executable name. Subtle differentiates between regular top-level windows and popup windows (which are owned by another top-level window) and notes when a user opens, closes, moves, sizes, or switches between windows. Mouse and keyboard input events are captured, as are focus-change events (our prior work has found that focus-change events can be an indicator of task engagement [11]). We also log the overall CPU usage and the CPU usage of each process (a somewhat general approach to differentiating between an idle or active process, such as an idle media player versus one that is currently showing a movie). We log the current audio volume and whether or not audio output is muted. Finally, we log whether a computer is plugged in, as this seems likely to be an indicator of whether a person is in a semi-permanent work area or a more temporary situation.

Subtle monitors the primary audio input device (typically a small microphone built into the case of a laptop computer), conducting several analyses of ambient audio. It seems intuitive that certain applications should behave differently in noisy environments, so we compute two volume-related statistics, using them to sense the magnitude and variation in the overall noise level. Our prior work on human interruptibility has found that nearby conversation is an indicator of social engagement [9, 10], so we compute several features to detect nearby speech. The first is the energy level in the frequency range of human voice. We also compute features based on the high zero-crossing rate ratio (HZCRR), the low short-time energy ratio (LSTER), and spectrum flux (SF), all presented by Lu *et al.* in their work on audio segmentation [28]. Lu *et al.* have shown that these features are effective for recognizing the difference between speech, music, and environmental sounds. They have also proven effective in our deployment of a context-aware instant messaging client [12].

WiFi-based location sensing is implemented using periodic scans for nearby access points, an approach proven effective by the Place Lab initiative [26]. Subtle logs the MAC address, network name, and signal strength of each detected access point. Subtle also logs which access point is currently in use. The inclusion of a location sensor may allow a model to be based on whether a person is in their primary work area (the most common set of access points in this person's history), in some other location that they frequent (a set of previously encountered access points), or in an unusual or new location (a set of access points that has not previously been encountered).

### Sensor Extensibility Support

While the provided sensors should be sufficient for many applications (they are a superset of those in our prior work on human interruptibility), other applications may benefit from custom sensing. Subtle supports two approaches to this. The first is based in Subtle's state and event readings. If an application has context it wants to provide for consideration by a model, or if a new sensor is being developed, it can provide Subtle with an XML-encoded reading. Subtle will treat this reading as if it were generated by our provided sensors: parsing it, applying the privacy policy, and storing it in our database for later analysis.

A second approach is appropriate when Subtle is integrated with a large existing system. For example, consider if an application has been built with the Context Toolkit [4], but the developer wants to add Subtle's support for model development and deployment. To address this type of situation, Subtle provides a level of abstraction surrounding database queries. The primary implementation uses timestamped name/value pairs stored in our database illustrated in Figure 2, but custom implementations can associate additional databases with Subtle. In the scenario described, a developer would provide a custom implementation that queries the existing Context Toolkit database. Subtle queries for timestamped reading values would then be routed through this implementation, allowing Subtle to learn from and execute models against the data contained in the Context Toolkit.

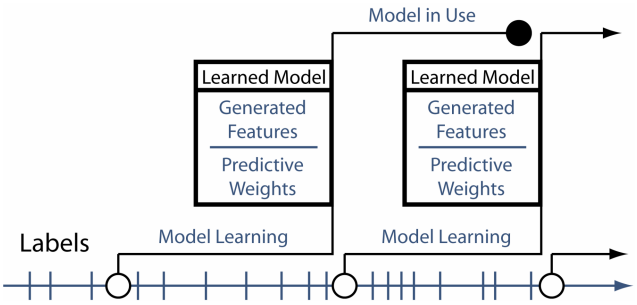


**Figure 3. As additional labels are collected, Subtle continuously learns updated sensor-based statistical models.**

## CONTINUOUS LEARNING OF PERSONALIZED MODELS

To address the problem that *it is hard for a static model to account for individual differences and unexpected situations,* Subtle supports continuous learning of personalized models using labels collected from an individual end-user. Figure 3 presents the lifetime of Subtle models. The bottom timeline represents the occasional collection of labels by an application. When Subtle begins its fully-automated learning process, it takes a snapshot of available labels (represented by the vertical transition from the unshaded circle). Subtle's learning process is then executed against that set of labels. When complete, the learned model begins to be used for live sensor-based estimates (represented by the second vertical transition). More labels will probably have been collected in the time taken to learn the model, so the process begins again with a new snapshot. When this new model is learned, it is promoted into use. The lifetime of the original model ends here, marked by a shaded circle.

The storage of sensor readings for later analysis by Subtle's model learner raises the important issue of potentially sensitive data in sensor logs. To address this concern, Subtle filters each sensor reading through a hash-based privacy policy before storing it in Subtle's database (as illustrated in Figure 2). Subtle's default privacy policy is based in the application of a one-way cryptographic hash to potentially sensitive strings. This masks the content of a string while still allowing Subtle to learn about meaningful values of the string (because the same string will always have the same hash). Subtle's default privacy policy decides whether a string is potentially sensitive based on how it was obtained (as opposed to being based on the content of a string). For example, the default privacy policy considers window titles to be potentially sensitive (because they could reveal logins or other indicators of a person's identity), but does not consider the active executable filename to be potentially sensitive. It therefore tokenizes and applies a one-way cryptographic hash to window titles, but stores the actual value of the active executable filename. This default policy provides an end-user with appropriate protection without requiring any effort of an application developer. Because some applications will desire weaker or stronger privacy policies, Subtle allows applications to define an arbitrary privacy policy, either by changing what readings are hashed or by providing a new implementation to apply an arbitrary transformation to collected readings.

A second issue raised by continuous learning is that it would be unacceptable to consume a large portion of a computer's processor while learning a model. Assigning the learning process a low priority might prevent it from interfering with other applications, but the process would still quickly drain the battery of a laptop computer. Subtle therefore manages its own processor usage. When a person is active on their computer (indicated by mouse or keyboard activity in the past 5 minutes) or when the computer is running from battery, Subtle limits the model learner to approximately 15 percent processor utilization. When both plugged in and idle, Subtle allows 80 percent processor utilization.

The final issue we discuss regarding continuous learning of personalized models is how an application should behave in the time before a reliable personalized model becomes available (including the time to collect labels and to learn the model). Based on our prior result that relationships with some sensed context are predictive for a large variety of office workers [9], we believe it will often be useful to provide a generic model for use until a personalized model is available. Subtle supports transitioning from a generic model to a personalized model by exposing information about the labels used to train each model and the estimated reliability of a model. An application can use this information to decide when it is appropriate to begin using a personalized model. An application might also decide not to use individual models, instead collecting labels to update one or more generic models available to application users.

## AUTOMATED FEATURE GENERATION AND SELECTION

While the collection of appropriate sensor readings is necessary for learning a model, it is generally not sufficient. Models instead need to be built from features that extract higher-level concepts from sensed context. For example, the exact value of a volume reading from an ambient audio sensor is probably less predictive than a Boolean feature capturing whether the reading is above a specific threshold. This feature might in turn be less predictive than one that captures whether the volume was above that threshold at any point in the past 30 seconds. Creating appropriate features is critical to successful machine learning, but *it is hard to determine what aspects of available context are most useful in a model*. Non-experts can be very unsure about how to create appropriate high-level features, and even experts are unlikely to manually craft the best possible features [29]. Because Subtle learns personalized models in a background process on an end-user's computer, we have the additional requirement that we generate appropriate features without human intervention in the learning process.

Subtle addresses this problem using a fully-automated and feature generation process based on each potential feature's type and history of values. Figure 4 shows an example of an automatically-generated feature for whether a person has been within range of a specific WiFi access point for 10 of the past 15 minutes. Two major abstractions are involved in the generation of such a feature. An *operator* creates a new potential feature by applying a transformation to an existing

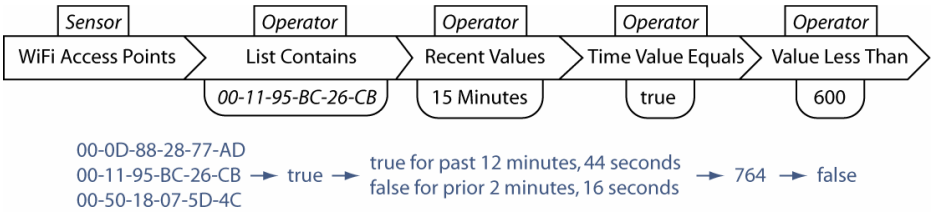


**Figure 4. An automatically generated feature for "Within range of a specific WiFi access point for 10 of the past 15 minutes." Because new features are automatically generated based on a feature's type and the history of values associated with that feature, these same operators can be used to create a feature like "Windows Media Player was open for 8 of the past 10 minutes."**

potential feature. Operators are added to features by a *generator*, each of which examines the type and history of values associated with existing potential features.

In the case of Figure 4, the low-level sensor reports a list of detected WiFi access points. Subtle's generators examine this to consider what transformations are appropriate. The *List Contains* generator matches features of type list, so it examines the history of values reported for the list and identifies list elements that occur with a frequency that may warrant further examination. For each such element, a new potential feature is generated by adding the *List Contains* operator parameterized with the value of the potentially interesting element. The resulting feature is Boolean, and so different generators are appropriate for iterating upon it. One such match is for *Time Value Equals*, which identifies a potentially interesting value and a timespan over which to check a feature's equality with the value. This yields a numeric feature, which is examined by a generator that computes the optimal information-theoretic threshold.

Because this process is based on the type of each existing feature and the history of values reported for that feature, as opposed to manually-specified semantics about what types of features should be explored based on what underlying sensors are being considered, Subtle can automatically apply existing operators to new sensors or apply new operators to existing sensors. In contrast, an approach based in defining what operators should be explored with what sensor would require the manual re-examination of every existing operator and sensor when creating a new operator or sensor. For example, consider starting from a list of open application executable names. Identifying *wmplayer.exe* as an interesting element of the list, the same operators can generate a feature capturing whether a person had Windows Media Player open for 8 of the past 10 minutes.

### Feature Generation

Subtle's extensibility allows arbitrary transformations in an operator and arbitrary exploratory processing of sensor data in a generator, but we also provide a set of operators and generators that efficiently explore features based on Subtle's current sensing library. We expect these operators and generators will also be effective with new sensors similar to those provided by Subtle. Our goal in providing these operators and generators is not a general learning algorithm (an extremely difficult artificial intelligence problem), but to make the development and deployment of sensor-based statistical models accessible to non-experts.

Subtle currently defines types for Booleans, numeric values, unprocessed strings, strings hashed by Subtle's privacy policy (that are therefore represented as a fixed-length byte sequence), strings containing an XML document, and lists of values. New types can be defined by implementing an interface with a comparison method and with methods for storing and retrieving a value from an XML stream. Space constraints prevent us from providing an exhaustive list of operators and generators based on these types, but this section presents some of the most interesting operators and generators included with Subtle.

Numeric values are compared to thresholds by two different generators, *Discretize* and *Value Less Than*. Both use the information gain statistic to find the information-theoretic optimal split point for numeric features. *Value Less Than* generates exactly one split for every existing numeric feature. *Discretize* uses Fayyad and Irani's method, recursively choosing split points according to information gain with the minimum description length principle (MDLP) providing a stopping criterion [7]. We use both of these approaches because the MDLP criterion can be somewhat difficult to satisfy. *Value Less Than*'s generation of a single split allows every numeric feature an opportunity to emerge as predictive. But a single split is not always optimal, so *Discretize* allows multiple split points when the MDLP criterion indicates that the result is very likely to be useful in a statistical model.

The *Value Equals* generator is applied to all types, as it is implemented using only the comparison method required of every type. It examines the values of an existing feature at the time of each label and creates a new Boolean feature for each value that occurs with a frequency that might warrant further examination. This generator, for example, can identify *WINWORD.EXE* as a potentially interesting value of *Active Application*. As discussed regarding Figure 4, the *List Contains* generator takes a similar approach to identifying potentially interesting elements of lists.

Several generators create potential features that examine the recent history of values for an existing feature. The *Most Recent Value* operator is useful with event-based sensors (such as analyses of GUI desktop environments), because a label is unlikely to occur in exactly the same instant as a potentially predictive event and it is useful to be able to look into the past for the most recent occurrence of an event. Other generators based on a recent history of values include *Min*, *Max*, *Mean*, *Median*, *Value Change Count*,

*Most Common Value*, *Time Value Equals*, and *Time Since Value Equals*. These and other history-based operators use a utility operator, *Recent Values*, to collect the history of values for a feature in a time interval (as seen in Figure 4). They also ensure that they do not add a *Recent Values* operator to a feature that already contains a *Recent Values* operator, thus preventing the unnecessary examination of features that are unlikely to be useful (such as the *Max* of the *Min* of the *Most Common Value* in the past minute).

### Feature Selection

Subtle's feature generation yields many potential features, most of which are not predictive and should not be used in a model. Subtle currently applies two filters: a *correlation* filter and an *optimal subset* filter.

The correlation filter uses computationally inexpensive heuristics to quickly reduce the number of features under consideration. Features that have too many different values are filtered, as are features for which a value occurs in an overly small percent of data. Filtering these features helps to prevent overfitting, wherein models mistakenly treat minor details of training data as important and therefore have an unnecessarily low reliability when applied to new situations. The filter then uses several measures of correlation to select from the potential features. A machine learning researcher manually crafting a solution could experiment to determine what notion of correlation is best for a particular dataset, but our fully-automated approach leads us to use several in parallel. Subtle computes the information gain, gain ratio, and symmetrical uncertainty of each feature relative to the labels [30, 35]. The filter selects the *n* best-correlated features for each measure (features are selected if they are in the top *n* for any of the measures, where the default *n* is 1000). Finally, the filter uses Yu and Liu's notion of predominance [35] to select a small number of features that are not in the top *n* but still provide predictive value distinct from the top features.

The optimal subset filter examines the remaining features with a best-first search in a wrapper-based feature selection process [25]. This search starts with an empty feature subset, adding and removing potential features until no change results in improvement (with limited support for backtracking provided by the best-first strategy). The utility of each feature subset is evaluated using a standard ten-fold cross-validation (dividing the data into ten folds and using each fold to test a model trained from the other nine) to estimate the area under the ROC curve (the area under an ROC curve is related to accuracy but avoids pitfalls of optimizing directly for accuracy) [8, 13, 14].

Wrapper-based feature selection requires a computationally inexpensive classifier. Subtle currently chooses between naïve Bayes [5, 27] and decision tree [32] classifiers. The decision is made during the wrapper-based selection of optimal features, based on which type of model yields a better score. This allows Subtle to adapt to the different types of data for which these classifiers are appropriate.

| Iterations Completed | Unique | Correlate | Optimal | Area Under ROC Curve | Accuracy |
|---|---|---|---|---|---|
| 1 | 97 | 96 | 0 | **.500** | . 585 |
| 2 | 1322 | 212 | 13 | **.633** | .636 |
| 3 | 8247 | 1061 | 27 | **.714** | .714 |
| 4 | 23952 | 1180 | 42 | **.751** | .733 |
| 5 | 33227 | 1347 | 25 | **.779** | .739 |
| 6 | 34801 | 1358 | 27 | **.787** | .760 |
| 7 | 34805 | 1358 | 27 | **.787** | .760 |
| 8 | 34805 | - | - | - | - |

**Figure 5. Summary of a model's iterative development, with how many features passed each filter in each iteration.**

### Iterative Feature Generation and Selection

The feature generation and selection processes just discussed are executed within an iterative process. Starting from the unprocessed low-level sensors, Subtle applies generators to create new potential features. These features are filtered, with the final filter selecting an optimal subset. Subtle then decides whether to continue creating new potential features. Iteration terminates after a pass in which generators do not create any new potential features or after five passes without any improvement in the scoring of the optimal features (based on the area under the ROC curve).

Figure 5 presents a summary of Subtle's iterative model development for a dataset presented later in this paper. We have bolded the area under the ROC curve as a reminder that iteration continues until it converges. This dataset contains 97 low-level event-based sensors. While 96 of these pass our correlation filter, they are too low-level to be of any benefit in a model. Applying our generators to the 97 low-level sensors creates 1322 unique potential features. A second iteration, applying generators to these 1322 features, yields 8247 potential features. As the number of potential features grows, the correlation filter quickly reduces the number under consideration. In this case, learning ends when the eighth iteration yields no new features.

## SUBTLE DISCUSSION

As discussed in our introduction, Subtle enables the development of applications that monitor, learn from, and respond to sensed context with just a handful of code. While we feel this is a clear demonstration that Subtle eases the development and deployment of sensor-based statistical models, a more formal evaluation is inherently difficult. Subtle enables research that would have been extremely difficult to pursue, but an evaluation based on a particular application or dataset provides only indirect insight into Subtle's utility [6]. This section examines Subtle by considering three issues. We first discuss Subtle's learning mechanisms with regard to two datasets collected in prior work. We then present two demonstration applications showing how little effort is required when using Subtle to develop and deploy sensor-based statistical models. Finally, we report on the adoption of Subtle by other researchers pursuing their own work, including two examples of deploying Subtle applications in field studies of how people use applications that include sensor-based statistical models.

### Model Learner Discussion

Subtle's current set of operators and generators has been informed by our work on sensor-based statistical models of human interruptibility [9]. In that work, we deployed a set of sensors in the offices of ten office workers with diverse responsibilities (the sensors were a subset of those provided by Subtle, plus motion detectors and contact switches). We collected interruptibility self-reports with an experience sampling method and found that participants considered themselves to be interruptible for 67.9% of collected labels. Using a traditional machine learning approach with manual construction of appropriate high-level features, we created and published a model with an accuracy of 79.5% [9].

We used this dataset, together with our experience in applying a traditional machine learning approach to it, to iteratively develop Subtle's core operators and generators. This provided a concrete dataset, with sensors similar to those provided by Subtle, for use in informally determining what types of operators yield good results. Applying Subtle's automated learner yields a model containing 40 automatically-generated features and an accuracy of 80.0%. Subtle's model is significantly better than human observers (A' = .856 vs. A' = .724, $Z = 7.13$, $p < .001$) [10] and performs as well as our previously published model based on manually crafted features ($\chi^2(1, 1981) = 0.09$, $p \approx .77$).

As a demonstration of these same operators working well with another dataset, we have applied Subtle's automated model learner to data collected from programmers working on a realistic programming task while responding to interruptions [11]. This dataset is based on low-level events logged within Eclipse, a modern development environment. Applying the operators developed through experimentation with our previous dataset, Subtle's model learner created a model with 27 automatically-generated features and an accuracy of 76.0%, significantly better than the 58.5% base for this data (A' = .787, $Z = 14.1$, $p < .001$). This provides an example of Subtle's operators working with well a different dataset, and Figure 5 shows that the performance of this model is due to the high-level features automatically created by Subtle. As can be seen in Iteration 1 of Figure 5, the 97 event-based sensors in the unprocessed dataset are too low-level to provide useful information to a machine learning algorithm. The machine learning algorithm does not begin to perform well until later iterations, after Subtle has automatically generated useful high-level features.

### Application Development Discussion

Having discussed Subtle's iterative feature generation, we now turn to two applications that demonstrate how Subtle eases the development of applications that use sensor-based statistical models of human situations. Initially presented in Figure 1, *Whistle* is an application that monitors when people mute or enable the audio of a laptop computer. It collects a label each time a person manually toggles their audio, learning a sensor-based statistical model that it uses to automatically mute and enable audio. This model is based on Subtle's provided sensors, so it can learn such concepts as "*mute audio when I am not at home*" or "*enable audio when Windows Media Player is active*."

Implemented in Java with a native library for querying and setting the operating system's audio mute flag, Whistle has 265 lines of substantive code (including assignments and invocations while excluding imports, variable declarations, constant definitions, etc.). Of this code, 110 lines are GUI-related (the system tray icon, its popup menu, and the notification in Figure 1). Another 100 are native code for querying and setting the operating system's audio mute flag. 27 lines are related to coordinating Java threading and timing. Only 16 are directly related to the use of Subtle. Of these, 6 are overhead (start and stop Subtle, classloader configuration). Another 4 lines are invoked every time a label is collected (connect to Subtle, create the label object, and provide it to Subtle). The final 6 are invoked every time the model is evaluated (connect to Subtle, obtain the model, evaluate the model, compare the result to action thresholds).

Although Whistle is quite simple, its approach to learning and the code it uses to employ Subtle are typical of many potential applications. It is straightforward to envision a similar application learning to select a person's preferred printer (perhaps based on location or the active application), to reduce a laptop's processor power consumption (scaling back the processor in contexts that historically do not result in a demand for processor time), or to adjust a computer's display resolution (perhaps because the projector in a commonly-used room cannot handle the normal resolution).

Figure 6 shows *AmIBusy Prompter*, an application very similar to BusyBody [19]. Implemented in 250 lines of substantive code, primarily for its GUI, AmIBusy Prompter collects interruptibility self-reports at random intervals. These are used to learn a model of interruptibility, and an application can then consider a personalized interruptibility model with as little as 6 lines of code (2 for overhead, 1 to connect to Subtle, 1 to obtain the model, 1 to evaluate the model, and 1 to compare to a threshold). Subtle thus makes sensor-based statistical models of human interruptibility accessible to the larger human-computer interaction research community, not just groups that have appropriate sensing libraries and access to machine learning expertise.

### Subtle Usage Discussion

Subtle is available at http://subtle.cs.washington.edu, and we made early versions available to several researchers. This section discusses two research projects conducted with Subtle [21, 33]. Researchers were given access to both Subtle and to the source code for example applications (including Whistle and AmIBusy Prompter). Because we were supporting real research, we responded to questions about the best strategies for using Subtle. However, we did not provide direct programming assistance. When preparing this paper, we asked these researchers to share their code.

In the first project, Subtle has been used to study how users develop mental models of intelligent systems [33]. These researchers deployed AmIBusy Prompter for three months

with four participants, creating personalized models of each participant's interruptibility. They then used Subtle to create continuously updating door-mounted displays of the interruptibility of the four participants. These displays were deployed for six weeks, and the researchers conducted a series of interviews with colleagues of the participants to study their mental models of the interruptibility estimates.

We consider this use of Subtle in a multi-month field study to be an important validation of Subtle's utility. Examining the source code shared by these researchers and discussing it with them, we found the expected pattern of copying code snippets from the source code of our example applications (such as a snippet for obtaining an estimate from a model). We also found the researchers implemented a cross-entropy algorithm for examining the importance of each individual feature in a specific estimate, an algorithm that goes well beyond the usage illustrated in our example applications. This algorithm required the researchers implement a new nil value type, iterate through the features contained in a model, and examine how a substitution of nil for the actual value of each feature affected the output of the model. This is very different from any functionality in our examples, and its implementation by these researchers is an interesting case of going well beyond reusing example code.

In the second project we discuss here, Subtle has been used in building a context-aware instant messaging client [21]. The instant messaging client developed by these researchers can share a person's interruptibility, location, and desktop activity with interested colleagues. These researchers are focused on privacy control and feedback mechanisms, so using Subtle allowed them to focus on these mechanisms rather than the sensing and machine learning algorithms. In another example of the deployment of an application built with Subtle, these researchers have deployed their instant messaging client with ten participants for two weeks.

Examining the source code shared by these researchers and discussing it with them, we again found a pattern of reusing snippets from the example applications we provided. But these researchers have also gone beyond the examples, in this case by directly accessing Subtle's stream of sensor readings and extracting context (such as information about the active window) for use by the instant messaging client. They chose to directly access Subtle's stream of sensor readings because the information they were using is filtered by the default privacy policy, as a history of active window titles may contain logins or other sensitive information. Subtle's flexibility in giving the application access to the underlying sensor stream allowed appropriate short-term use of this information while maintaining the necessary privacy policy in the logs collected by Subtle.

### Limitations

Subtle currently cannot model continuous labels (such as response times), and no support is currently provided for more advanced users to choose between different machine learning algorithms. We are currently addressing this limitation by integrating Weka [34] into Subtle. Subtle will then be able to automatically choose a regression-based algorithm when given continuous labels. Advanced users will also be able to provide parameters directly to Weka.


**Figure 6. AmIBusy Prompter learns a personalized model of interruptibility that can be used with just 6 lines of code.**

Learning a model from a hundred or more labels collected by AmIBusy Prompter can currently take more than a day of CPU time. Much of this is due to simple algorithmic inefficiencies that we are currently addressing in a refactoring of Subtle, but there also exists a fundamental tension between exploring additional features and the time needed to explore them. We have currently crafted our set of operators to explore a meaningful space of potential features, but a poorly-conceived extension could cripple Subtle by introducing a computationally intractable set of features. We are therefore investigating a more structured approach to Subtle's generators. By examining generators that work well in Subtle, we hope to introduce additional support for well-behaved generators while also removing or at least detecting poorly-behaved or ineffective generators.

## CONCLUSION

We have presented Subtle, a toolkit that removes many obstacles to developing and deploying applications that use sensor-based statistical models of human situations. Subtle includes an extensible sensor library targeted at laptop computers, provides continuous learning of personalized models, and is based on a fully-automated approach to iterative feature generation. We have discussed Subtle's model learner in the context of two datasets, presented two examples of using sensor-based statistical models in applications with just a handful of code, and discussed some experiences with other researchers using Subtle in their own work. By supporting the development and deployment of sensor-based statistical models of human situations, Subtle helps focus human-computer interaction research on applications and datasets, instead of the difficulties of collecting sensor data and learning statistical models. Subtle thus enables future research into how human-computer interaction can benefit from sensor-based statistical models of human situations.

## ACKNOWLEDGMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010, and by the National Science Foundation under grants IIS-0121560 and IIS-0325351.

## REFERENCES


1. Bailey, B.P., Adamczyk, P.D., Chang, T.Y. and Chilson, N.A. (2006) A Framework for Specifying and Monitoring User Tasks. *Computers in Human Behavior*, *22*(4). 658-708.
2. Begole, J.B., Tang, J.C. and Hill, R. (2003) Rhythm Modeling, Visualizations, and Applications. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2003)*, 11-20.
3. Dey, A.K., Mankoff, J., Abowd, G. and Carter, S. (2002) Distributed Mediation of Ambiguous Context in Aware Environments. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2002)*, 121-130.
4. Dey, A.K., Salber, D. and Abowd, G.D. (2001) A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal*, *16*(2-4). 97-166.
5. Duda, R.O. and Hart, P.E. (1973) *Pattern Classification and Scene Analysis*. John Wiley and Sons.
6. Edwards, W.K., Bellotti, V., Dey, A.K. and Newman, M.W. (2003) Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*, 297-304.
7. Fayyad, U.M. and Irani, K.B. (1993) Multi-Interval Discretization of Continuous Valued Attributes for Classification Learning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1022-1027.
8. Fogarty, J., Baker, R.S. and Hudson, S.E. (2005) Case Studies in the use of ROC Curve Analysis for Sensor-Based Estimates in Human Computer Interaction. *Proceedings of Graphics Interface (GI 2005)*, 129-136.
9. Fogarty, J., Hudson, S. and Lai, J. (2004) Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*, 207-214.
10. Fogarty, J., Hudson, S.E., Atkeson, C.G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J.C. and Yang, J. (2005) Predicting Human Interruptibility with Sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *12*(1). 119-146.
11. Fogarty, J., Ko, A.J., Aung, H.H., Golden, E., Tang, K.P. and Hudson, S.E. (2005) Examining Task Engagement in Sensor-Based Statistical Models of Human Interruptibility. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*, 331-340.
12. Fogarty, J., Lai, J. and Christensen, J. (2004) Presence versus Availability: The Design and Evaluation of a Context-Aware Communication Client. *International Journal of Human-Computer Studies (IJHCS)*, *61*(3). 299-317.
13. Hand, D.J. and Till, R.J. (2001) A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, *45*(2). 171-186.
14. Hanley, J.A. and McNeil, B.J. (1982) The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, *143*. 29-36.
15. Ho, J. and Intille, S.S. (2005) Using Context-Aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*, 909-918.
16. Horvitz, E. and Apacible, J. (2003) Learning and Reasoning about Interruption. *Proceedings of the International Conference on Multimodal Interfaces (ICMI 2003)*, 20-27.
17. Horvitz, E., Jacobs, A. and Hovel, D. (1999) Attention-Sensitive Alerting. *Proceeding of the Conference on Uncertainty and Artificial Intelligence (UAI 1999)*, 305-313.
18. Horvitz, E., Kadie, C., Paek, T. and Hovel, D. (2003) Models of Attention in Computing and Communication: From Principles to Applications. *Communications of the ACM*, *46*(3). 52-59.
19. Horvitz, E., Koch, P. and Apacible, J. (2004) BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*, 507-510.
20. Horvitz, E., Koch, P., Kadie, C.M. and Jacobs, A. (2002) Coordinate: Probabilistic Forecasting of Presence and Availability. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, 224-233.
21. Hsieh, G., Tang, K.P., Hong, J.I. (2007) The Design and Evaluation of Privacy Controls and Feedback Mechanisms for Contextual Instant Messaging. *Submitted for Review.*
22. Hudson, S.E., Fogarty, J., Atkeson, C.G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J.C. and Yang, J. (2003) Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*, 257-264.
23. Iqbal, S.T., Adamczyk, P.D., Zheng, X.S. and Bailey, B.P. (2005) Towards an Index of Opportunity: Understanding Changes in Mental Workload During Task Execution. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*, 311-230.
24. Kern, N. and Schiele, B. (2003) Context-Aware Notification for Wearable Computing. *Proceedings of the IEEE International Symposium on Wearable Computing (ISWC 2003)*, 223-230.
25. Kohavi, R. and John, G.H. (1997) Wrappers for Feature Subset Selection. *Artificial Intelligence*, *97*(1-2). 273-324.
26. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G. and Schilit, B.N. (2005) Place Lab: Device Positioning Using Radio Beacons in the Wild. *Proceedings of the International Conference on Pervasive Computing (Pervasive 2005)*, 116-133.
27. Langley, P. and Sage, S. (1994) Induction of Selected Bayesian Classifiers. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 1994)*, 399-406.
28. Lu, L., Zhang, H. and Jiang, H. (2002) Content Analysis for Audio Classification and Segmentation. *IEEE Transactions on Speech and Audio Processing*, *10*(7). 504-516.
29. Markovitch, S. and Rosenstein, D. (2002) Feature Generation Using General Constructor Functions. *Machine Learning*, *49*(1). 59-98.
30. Mitchell, T.M. (1997) *Machine Learning*. McGraw-Hill.
31. Nagel, K.S., Hudson, J.M. and Abowd, G. (2004) Predictors of Availability in Home Life Context-Mediated Communication. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*, 497-506.
32. Quinlan, J.R. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
33. Tullio, J., Dey, A.K., Chalecki, J. and Fogarty, J. (2007) How it Works: A Field Study of Non-Technical Users Interacting with an Intelligent System. *ACM Conference on Human Factors in Computing Systems (CHI 2007)*, To Appear.
34. Witten, I.H. and Frank, E. (1999) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.
35. Yu, L. and Liu, H. (2003) Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. *The International Conference on Machine Learning (ICML 2003)*, 856-863.