# Machine Learning for Activity Recognition

**Jianqiang Shen**

Dearborn102, School of EECS
OSU, Corvallis, OR97331
shenj@cs.orst.edu

## Abstract

This paper surveys the activity recognition task from a machine learning perspective. I give a definition of this problem, and I classify different activity recognition problems into two categories. I show the activities can be hierarchical, and based on such hierarchies I synthesize a language to describe activities. I give a general criteria set to evaluate activity recognition methods. I summarize some off-the-shelf machine learning methods for activity recognition and evaluate them based on this criteria set. Finally, I discuss some methods that I believe can improve the activity recognition performance.

## 1 Introduction

The need for personal cognitive assistants is continuously increasing. In the United States, the number of people over the age of 65 will double between now and 2030 to 69.4 million [24]. Historically, 43% of people over the age of 65 enter a nursing home for at least one year. Automatic caregivers will help them both physically and psychologically.

At the same time, as the computer and Internet become more and more popular, computer crimes have become a serious problem [21]. To make important systems safer, log data must be monitored all the time. Because manual monitoring is tedious, monitors may become distracted and miss significant events. We therefore need some automatic monitoring systems.

All these applications involve activity recognition: they observe a sequence of measurements, try to guess the goal of the subjects, and respond to it. Activity recognition will play an important role in our lives.

In Computer Science, perhaps the earliest research related to activity recognition is *plan recognition* [50, 39]. The problem of plan recognition is to induce the plan of action driving an agent's behavior, based on partial observation of its behavior up to the current time [51]. Deriving the underlying plan can be useful for many purposes – predicting the agent's future behavior, and enhancing intelligent user interfaces.

In medical domains, devices have been used to assist people with cognitive disabilities (such as learning disabilities and traumatic brain injury) in accomplishing *Activity of Daily Living* (*ADL*) for close to 20 years. Most often these devices have most often been referred to as cognitive orthoses or cognitive prostheses [42]. However, one drawback of these devices is that they require the user's explicit feedback to indicate which step has been finished, such as pressing a button. Sometimes, it's impossible to achieve this requirement.

People didn't begin to integrate machine learning into activity recognition until recently. Some commercial applications have been developed based on machine learning methods. The *Office Assistant* in *Microsoft Office* [27] is perhaps the most famous one among them. Also, some important workshops were held: the AAAI-02 Workshop "Automation as Caregiver", the NIPS-03 Workshop "Machine Learning Meets the User Interface", and the ICML-04 Workshop "Physiological Data Modeling – A Competition". An important conference is called International Conference on Ubiquitous Computing. Ubiquitous computing tries to help people with computer technology in the physical world. The Sixth UbiComp (*UbiComp2004*) was held September 7-10, 2004. This annual conference provides the premier forum in which to present research results in all areas relating to the design, implementation, application, and evaluation of ubiquitous computing technologies.

For a detailed survey about machine learning in activity recognition, please refer to [33, 42, 24].

In this paper, I formalize the task of activity recognition and give a criteria set. I summarize off-the-shelf learning methods in activity recognition and evaluate their performance. In the end of this paper, I will discuss some possible methods that could improve the performance of activity recognition.

## 2 Formalizing Activity Recognition

Normally, activity recognition will observe a sequence of measurements (e.g. physical measurements, user-interface interactions). It tries to guess the goals of the subjects, responds to them accurately and promptly. According to

its goals, we can classify the problem into two categories: *activity monitoring* and *activity prediction*.

Activity monitoring tries to ensure that a normal activity sequence is being executed. The goal of activity monitoring is to issue alarms when abnormal things happen. A typical example is intrusion detection [3]. The computer system continuously observes the user's commands and analyzes them. If it believes some unusual pattern happens, it will report an alarm to administrators. Other examples include cell phone fraud detection [21] and driving monitoring [51]. Let $D = (d_{t-k}, \ldots, d_{t-1}, d_t)$ be an ordered set of observed data, where $k$ is 0 if we ignore the temporal relationship and only consider current observation; the goal of learning in activity monitoring is to give a mapping:

$$(d_{t-k}, \ldots, d_{t-1}, d_t) \rightarrow S_t$$

Where $S_t$ is a binary variable, indicating whether an abnormal activity is happening. In probabilistic models, it should give the probability that an abnormal activity is happening.

Activity prediction tries to guess the goal of users and predict which action will be executed next. Most of the time, it will try to assist users by doing some repetitive jobs or giving some clues. Such assistance is based on predictions of users' future actions. A typical example is a programming assistant, which tries to understand what users are doing [53]. If it believes users are going to do some repetitive things, it will tell users that these repetitive things can be done automatically. If users accept such an option, it will successively perform these actions. Other examples include the Office Assistant [27] and automatic caregivers [42]. Instead of simply answering with yes or no in activity monitoring, it should figure out what will be the most likely sequence of future actions. Formally, the goal is to give a mapping:

$$(d_{t-k}, \ldots, d_{t-1}, d_t) \rightarrow (d_{t+1}, d_{t+2}, \ldots, d_{t+m})$$

Depending on the application, $m$ will range from 1 (only predicting the next action) to $\infty$ (predicting the entire future situation). Of course, in probabilistic models, activity prediction should give the probability for different future action sequence.

### 2.1 Hierarchy of Activities

A complex activity may consist of several simple activities. For example [9], the activity *washing-hands* consists of *turning-on-water*, *using-soap*, *rinsing-hands*, *turning-off-water*, etc. At the same time, *turning-on-water* is also an activity, consisting of *holding-faucet*, *moving-clockwise*, *leaving-faucet*, etc.

Generally, we can consider the activity in three levels: cognitive level, functional level and executive level. Activities in the cognitive level reflect the mental status of the user (e.g., is the user interruptible?) Such activities may include some functional activities. At the same time, in order to do a functional activity, the user should execute some physical activities, which we will call executive activities. For example, if the activity of the user is interruptible, maybe he is programming or sleeping. Also, as a functional activity, programming may consist of some executive activities, such as typing and mouse moving. An example is given in Figure 1.

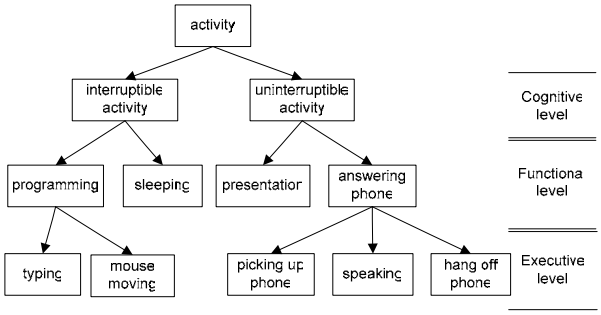



Figure 1: An example of Activity Hierarchy.

Most of the time, the observations of sub-activities are called *events*. For the sake of clarity, I will use this terminology in the following.

Typically, systems will observe the sub-activities and guess which sup-activity the user is doing. Based on this guess, systems can figure out the user's goal or plan, and thus can predict what sub-activities will possibly be executed next. For example, if we observe that a person picks up a phone and begins speaking, we will say he is answering a phone call. Thus, we will predict he will hang up the phone next.

It is very important to decide the level of activity based on which you make inferences. It belongs to the big problem – feature design. Considering the answering-phone example, we can conclude that people make inferences based on high level observations, such as picking up phone. Actually, picking up phone also consists of hand-move-down, grasping, hand-move-up, etc. But people don't make inference based on such primitive observation. Instead, they will abstract such observations into some high level activities, and then make inferences.

Another example is the Lumière project [27]. In this project, the direct observations are atomic events, such as mouse and keyboard actions, the status of data structures in Excel files, etc. Instead of inferring directly based on these atomic events, Lumière transforms them into higher level events, such as "menu surfing", "mouse meandering", and "menu jitter". Lumière built a Bayesian user model based on these modeled events.

This gives us a hint: it will be very helpful if we can abstract the activities into some higher level activities. For example, when we model the problem with a Bayesian Networks, the variable node will be reduced greatly if we use the abstracted activities. Also, such variables are more natural and contain more meaningful information. We can imagine the inference accuracy will also be much better. In the training or learning period, we transfer the observations into higher activities (most of the time, we

only need to do some pattern search, which is fast). Based on this abstracted information, the computation complexity will be reduced even exponentially.

### 2.2 A Language to Describe Activities

Because of the benefits previously discussed, there have been some efforts to describe activities [27, 31, 53, 56]. If we can define an activity clearly (what sub-activities it includes, in what order, etc.), we can convert low level activities into high level activities.

Based on the work of Horvitz [27] and Weiss [56], I proposed a language to describe activities. With this language, we can describe high level activities easily. This language provides the following primitives:

- "?" is a single-match primitive, which matches any single event value. So A?D matches ACD.
- "*" is an all-match primitive, which matches any number of events. So A*D matches ABCD.
- "/" is an "OR" primitive, which allows one of the events to occur. So A/B matches A or B.
- "+" is an addition primitive, which means one or more such event occurs. So A+ matches A, AA, AAA, and so on.
- "|" is an unordered primitive, which allows events in any order. Also, it is commutative, so A|B|C will match ACB, BAC, CBA, etc.
- "(" and ")" are constraint primitives, which mean the content between them should be considered together. So (AB)/C match AB or C, while AB/C match AB or AC.
- ":" is a time duration primitive, which means the events occur in a definite time interval. So, (AB):5 means events A and B occur in 5 minutes.

This language enables flexible definition to be constructed. For example, "*3 or more A events occur, then C or D event occurs in one hour*" can be represented by AAA+(C/D):60. We can also represent "*besides other events, at least 3 A and 2 B events happen in 10 minutes*" by (A*|A*|A*|B*|B*):10.

We can define the activity manually. This can encode rich expert knowledge in the learning process. It will definitely improve the inference accuracy. Another method is applying the generic algorithm to learn the activity pattern [56], which will be discussed in the following section.

### 2.3 A Criteria Set

Activity recognition consists of observing a sequence of measurements (e.g., transactions, physical measurements, user-interface interactions) and recognizing when a particular activity takes place. There have been many applications in this domain. To evaluate their result, we need some criteria. As in other domains, computation speed and prediction accuracy will be considered. But there will be some specific requirements.

First, about the speed, we should pay more attention in inference speeds. As we know, most of these applications take observations, guess the user's goal, and predict the user's next action. The inference should be on-line and promptly. Otherwise, the result will be less meaningful, even meaningless. For example, considering the automatic caregiver problem, let's assume ordinary inferences will take more than 1 minute (because of the huge number of observations that are possible). Then, this caregiver may be useless: after it figures out the user is washing hands, the user may have finished washing and began another activity.

As we know, Bayesian networks are expressive learning models. They can naturally model uncertainty. This is the reason they are popular in activity modeling. But, because exact inference in Bayesian networks is NP-hard [52], people have to put a lot of efforts into simplifying the BN structures in order to speed inferences. It's also true for dynamic Bayesian networks. Sometimes, people even have to give them up. For example, the *Office Assistant* gave up using dynamic Bayesian networks to infer users' activity [27, 26]. Rather, the system employed a small event queue and considered only the most recent events.

In the meantime, training speeds may not be crucial: most of these applications are idle during specific times. We can accumulate the data and do batch training. For example, at night we will not use the automatic caregiver, so the system can do some training at night even if it's time consuming.

Second, about the accuracy, we should take into account the goal of activity recognition: its overall goal is to take automated actions to optimize the user's expected utility based on their recognitions. Let's introduce two definitions: the *expected cost of delayed action* (*ECDA*) is the difference in the expected utility of taking immediate ideal action, and delaying the ideal action until some future time; the *expected cost of annoyance* (*ECA*) is the expected cost when the systems try to help users while the prediction is wrong [28]. A good activity recognition system should minimize the sum:

$$ECDA + ECA \tag{1}$$

Let *TP*= the number of true positive examples, *TN*= the number of true negative examples, *FP*= the number of false positive examples, *FN*= the number of false negative examples, *FPR* be false positive rate, and *FNR* be false negative rate, we will have:

$$FPR = \frac{FP}{FP + TN}$$
$$FNR = \frac{FN}{FN + TP}$$

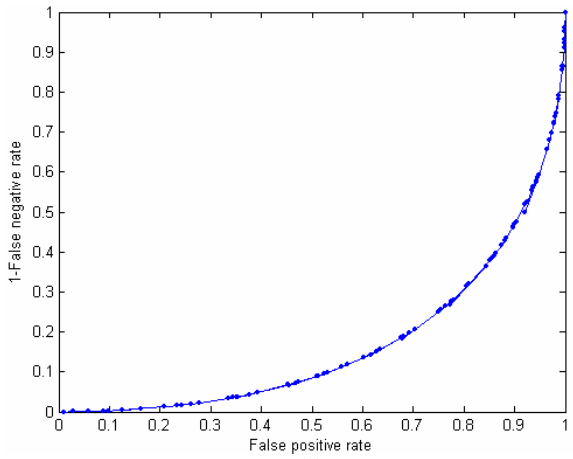
In most cases, *ECDA* corresponds to false negative rate, and *ECA* corresponds to false positive rate. Then, formula(1) will be equal to:

$$FNR + \lambda \bullet FPR \tag{2}$$

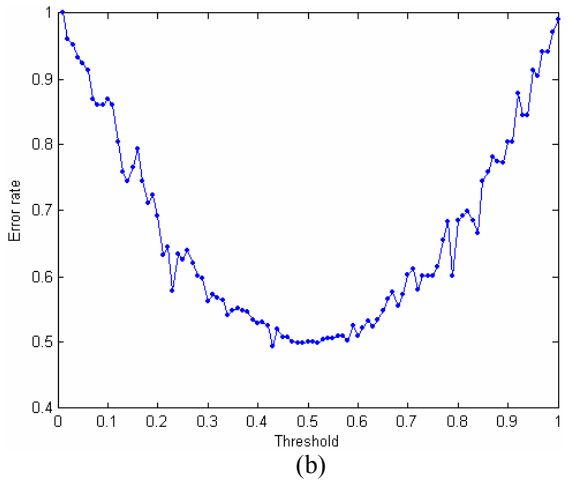
where $\lambda$ is the parameter that controls the costs of FPR relative to the costs of FNR. Poor "help" could be quite costly to users, so $\lambda$ may be greater than 1 in most cases.

Then, the Receiver Operating Characteristic (ROC) curve will be a good solution here. The standard ROC curve plots the false positive rate on the *X* axis and the 1-false negative rate on the *Y* axis [5].

I give a simple example of the ROC curve here: we sample 100 data points ($x_i$, $y_i$) where $x_i = i/100$, and $y_i$ is the class label which is 1 with probability $x_i$ , -1 with probability 1-$x_i$ . We consider a simple threshold classifier $f(x,\theta)$ here: if $x > \theta$ , we classify $x$ as positive, otherwise negative. We change $\theta$ from 0.01 to 1, increasing by 0.01 each time. Repeating this experiment for 100 times, we plotted the average error rate and the ROC curve in Figure 2.




(a)




(b)

Figure 2: (a) is the ROC curve, which plots (1–false negative rate) against the false positive rate; (b) plots the overall error rate against threshold $\theta$.

If the cost of false positive and false negative errors are the same, we will figure out the optimal $\theta$ is 0.5 from the error rate curve. But, if they are not the same, saying, cost=*FNR*+*R* • *FPR*, then the error rate curve can't help us at all. In this situation, the ROC curve is important: the best point is the point on the ROC curve tangent to the line with slope *R* that has the smallest intercept on the *FNR* axis.

Thus, a practical activity recognizer should make inferences in time, and have a low value of *ECDA*+*ECA* (this can be evaluated as *FNR*+λ • *FPR* sometimes). In the following section, we will evaluate methods based on these two evaluation functions.

# 3 Machine Learning Methods for Activity Recognition

As we mentioned before, there have been many applications for activity recognition. A lot machine learning methods have been tried, including decision trees [32, 53, 7], neural networks [12, 42, 7], genetic algorithms [56, 8], probabilistic models [27, 26, 25, 28, 3, 45, 47, 51], SVMs [11], etc. In most of the problems, the observations are uncertain. Because of this, probabilistic models are the most popular. People have tried naïve Bayesian models [25], Bayesian networks [27, 51], dynamic Bayesian networks [3, 28, 45] and other variations [13, 26, 27].

## 3.1 Data Preprocessing

Most of the time, the direct observations are not suitable for learning. They may contain some obvious outliers, the different attributes may be in different scales, and there may be too many unimportant attributes. To get a reasonable learning result, we should deal with noise, normalize the attributes and design learning features. Among these procedures, designing learning features is the most important. Much of the success of machine learning applications can be traced to careful engineering of the input features [18]. By designing features, we can even transform a temporal problem into an ordinary problem, thus we can simplify the learning task greatly. Data preprocessing is especially important to tasks related to physical measurements, such as BodyMedia [7, 8].

Reducing noise and normalizing attributes are trivial problems. There are mature methods to deal with them, but we lack automatic methodology for handling feature design. To make systems effective, the data analyst must carefully design the set of features. This is a time-consuming process, and there are relatively few data analysts who can do it. There are also very few software tools available to support this activity. Existing off-the-shelf machine learning systems do not provide any way to incorporate background knowledge except through defining the input features [18]. We'll focus on feature design here.

To get features more suitable for learning, we should apply the domain knowledge. Some existing attributes may make excellent features, but it is typical in machine learning applications to construct new features by a) aggregating existing attributes (e.g., over temporal and spatial scales) to reduce noise and improve statistical power, or b) transforming attributes (e.g., by Fourier or wavelet transforms and principal component analysis) to

enhance pattern detection [18].

An example is the Lumière project [27]. In this project, Horvitz et al. built an events system to establish a fluid link between low-level, atomic events and the higher-level semantics of user action, which they employed in user models. These modeled events, such as "menu surfing", are more suitable for learning than atomic events (e.g., mouse clicking, keyboard inputting, etc).

Another example is BodyMedia [7, 8]. In this problem, the armband could be worn or removed at the subject's discretion, and each wearing of the armband produced a *session* of sensor data. This problem is essentially temporal, since the observation is a sequence of measurements which are temporally correlated. But most of the attendees applied some feature design methods to get an ordinary supervised learning problem. We should notice the characteristics are the same across a session, so a naïve method uses the mean sensor values across the session as the features. A more sophisticated method is called the *histogram approach* [7]: the value range of every sensor in the session is divided into 50 buckets. For each sensor, they then devise 53 features: minimum of the sensor value, max value, mean value, and the number of data points falling into each of the 50 buckets.

### 3.2 Decision Trees

A decision tree is a hierarchical model implementing the divide-and-conquer strategy [52]. A decision tree is composed of internal decision nodes and terminal leaves. Each decision node $n$ implements a test function $f_n(x)$ with discrete outcomes labeling the branches. Given an input, each node applies a test and one of the branches is taken depending on the outcome. This process starts at the root and is repeated recursively until a leaf node is reached.

A decision tree can be translated into a logical formula for each class. If you take a single path down the tree, it can be translated into a conjunction of conditions, and the conjunctions for paths to leaves of the same class can be combined disjunctively. Decision trees can represent any logical formula.

Because decision trees are fast and have good expression power, they have been tried in activity recognition [32, 53, 7]. We will show how to use decision trees by the work at Carnegie Mellon University [32].

In Carnegie Mellon, a team has recently performed a Wizard of OZ study in an attempt to predict whether the users' activities are interruptible. In the study, the users' actions are recorded by a camera. Also, the users are intermittently asked feedback about the interruptibility. Researchers coded features by hand. In this way, each video interval was turned into a coded event, such as "speaking", "writing", "sitting", "interacting with keyboard", etc.

Based on this coded information, they also derived a number of variant sensors that captured recency and density effects. Some derived features are:

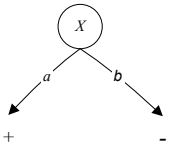
- Event occurred in the 15 second interval immediately around the self-report sample (***Imm***)
- Event occurred in every 15 second interval for 1 minute prior to the sample (***All-1***)
- Event occurred in at least one interval in the 1 minute prior to sample (***Any-1***)
- Event occurred in every interval in the five minutes prior to the sample (***All-5***)
- Event occurred in at least one interval in the 5 minutes prior to the sample (***Any-5***)
- The number of intervals in which the event occurred in the five minutes prior to the sample (***Count-5***)

Overall, they obtained observation values corresponding to a set of 128 direct or derived simulated sensors. In this way, they also change a temporal problem into an ordinary supervised learning problem.

They tried several learning methods and reported that decision trees have the best result. They believe it's because there is a strong and unambiguous feature (talking) that provides a very good initial split.

Decision trees are one of the fastest learning methods, thus, inference speed will be very satisfying. But, their accuracy will be a problem. First, they are not suitable for modeling the uncertainty of the problem. For example, when we are watching a person near a car, we are not sure he is locking the car or opening the car. Instead, we say: with a probability 0.6 he is opening the car, and with a probability 0.4 he is locking the car. If we watch he leaves the car later, we can be sure he was locking the car at that time. To model such problems, decision trees have to store probabilities in leaves. This is unnatural and can't reflect the causal relationships between variables.

Another problem comes from decision trees' inductive bias, a preference for the most general hypotheses [53]. Assuming we have data ($X$=$a$, $Y$=$b$, +), ($X$=$a$, $Y$=$c$, +), ($X$=$b$, $Y$=$d$, -), the learned tree will be like:




This tree will treat any point with $X$=$a$ as a positive example. This is too general considering the small number of training examples.

Too general hypotheses will match too many situations and raise too much positive predictions. For example, "*if users are static, give him a clue*" will always try to give users clues. As we mentioned, poor "help" could be quite costly to users. Decision trees lack a good mechanism to control generality and specialty. Thus, the *expected cost of annoyance* (*ECA*) will be very large.

For the above reasons, seldom decision trees are not very popular in real world applications.

### 3.3 Neural Networks

Neural networks attempt to model the operation of the brain using mathematics [52]. Each node, or neuron, in the network does a simple arithmetic operation, and gives the result to all of its successor nodes. When arranged in the appropriate network topology, arbitrary functions from the input to the output can be learned.

The simplest neural network consists of some input nodes and one output node, with all the input nodes connected directly to the output nodes. These are known as perceptrons, and have limited representational power, due to their simplicity. In a perceptron, the output is usually some function of the weighted sum of the inputs. That is, the output node computes $g(\sum_j w_j * x_j)$ over $j$ inputs and weights. $g$ is the activation unction, e.g. a step or sign function.

For representing complex relations, it is common to see networks with one or more hidden layers, which are neither inputs nor outputs, but intermediaries between them. It has been shown that with one layer of a large enough number of hidden units, any continuous function of the inputs can be represented, and with two hidden layers, discrete functions can be realized as well. Each node in a hidden layer can be seen as a perceptron, and computes the above equation over its own inputs, with its own weights.

The robust ability to classify new data makes NN very popular in real domains, especially computer vision. We will show how to use NN in activity recognition by an intelligent caregiver for handwashing [42].

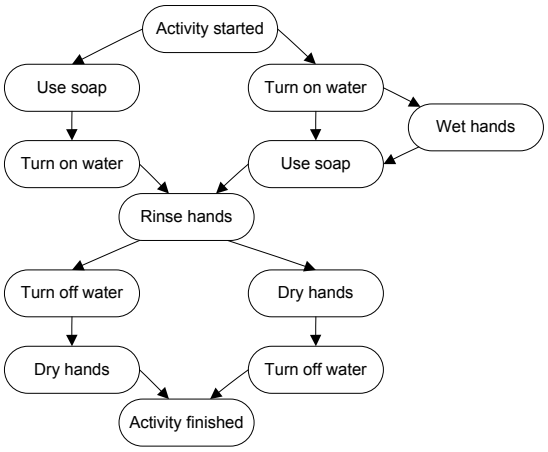



Figure 3: Acceptable sequences of steps required to complete the *handwashing*. Note wetting hands is considered optional in the prototype as liquid soap is used

In their prototype, they use color-based tracking software to follow the user's hand position through a camera mounted over the sink as the user performed handwashing activity. Figure 3 depicts six steps of handwashing and the various alternative pathways the user could correctly wash their hands.

A NN is used to learn which steps correspond with the various inputs from the environment. The inputs are abstracted from the camera information. The network classifies these inputs into step identification numbers.

In this way, they change the temporal problem into an ordinary supervised learning problem. Thus, it produces an ambiguous result: the position of the user's hand may not be uniquely related to a specific step. For example, the steps of turning the water on and turning it off have the same spatial coordinates. The only thing that distinguished these steps from each other is their positions in the overall plan. The system applies these rules by using prerequisites for each ambiguous step. For the example of turning the water off, a rule can be programmed that basically says "in order for the step being completed to be interpreted as turning the water off, the step of turning the water on must have already been completed". Using this rule, the system then searches a vector that keeps track of all of the completed correct steps by the user and sees if this prerequisite exists. If the prerequisite has not been completed, the system changes the initial step identification number to the one that is causing the ambiguity, in this case, turning the water on.

In most cases, neural networks can guarantee the inference speed. We can control the complexity of networks easily, thus tune the inference speed.

The problem is its accuracy. It will have high values of both the *expected cost of delayed action* (*ECDA*) and the *expected cost of annoyance* (*ECA*). First, neural networks are very sensitive to irrelevant inputs. This means we should be very careful about the feature design.

Second, neural networks are bad at handling with missing values, while we can't guarantee everything is observed in real domains. This may be the worst part of neural networks. In the handwashing project, they have to assume full observability of its washroom environment. This simplification does not account for inherent uncertainty in step identification introduced through factors such as instrumentation noise and obscured views. This is exactly the reason they are trying to upgrade the system to a *partial observable Markov decision process* (*POMDP*) based system [9].

### 3.4 Genetic Algorithms

Genetic algorithms depend on randomness to take decent solutions and make better ones [52]. This is done by combining solutions in such a way that the resulting solution may be better than the initial ones. Once a new set of solutions are created, the best are taken, and the process is repeated. This is analogous to Darwinian evolution, or survival of the fittest. In each generation, the best solutions survive, and reproduce, to make the new generation, which is once again tested for survival. After a pre-set number of generations, the algorithm terminates, and the best solution is returned.

Genetic algorithms include four key elements: the fitness function, individual representation, selection strategy and

reproduction strategy.

The fitness function depends on the problem, but in any case, it is a function that takes an individual as input and returns a real number as output. In the "classic" genetic algorithm approach, an individual is represented as a string over a finite alphabet. The selection strategy is usually randomized, with the probability of selection proportional to fitness. That is, if the individual *X* scores twice as high as *Y* on the fitness function, then *X* is twice as likely to be selected for reproduction as is *Y*. Reproduction is accomplished by cross-over and mutation. Cross-over means that offspring will get some parts from one parent, and the rest from another parent. Mutation means the offspring may get altered value with some probability. We will show how to use genetic algorithms in activity recognition by a rare activity predictor [56].

In this project, the data is a time-ordered sequence of events. Each target event, $X_t$, occurs at time *t*. The problem is to learn a prediction procedure *P* that correctly predicts the target events. Thus, *P* is a function that maps an event sequence to a Boolean prediction value.

They combined *Recall* (the percentage of target events correctly predicted) and *Precision* (the percentage of predictions that are correct) as the fitness function. Essentially, it's a derivative of ROC analysis.

The trick is how to represent pattern individuals. Here, they used a language similar to what we described in Section2.2. A *prediction pattern* is a sequence of events connected by *ordering primitives* that define sequential or temporal constraints between consecutive events. A *prediction pattern* matches a sequence of events within an event sequence if a) the events within the prediction pattern match events within the events sequence, b) the ordering constraints expressed in the prediction pattern are obeyed, and c) the events involved in the match occur within the pattern time duration. For example, if the *prediction pattern* is "A*C" and the sequence to be estimated is "ABC", we will say a target activity is happening.

They use a steady-state GA, where only a few individuals are modified in each "iteration", because such a GA is believed to be more computationally efficient than a generational GA when the time to evaluate an individual is large. With the above fitness function and individual representation, they applied GA to the problem. According to their report, GA has a fast converge speed, and the overall performance is much better than C4.5, FOIL, etc.

The inference speed is not a problem here: most of the time, the learned classifier only needs to do some simple computation. For example, in this project, the learned classifier compared the learned pattern with the estimated sequence. This is a string comparative operation, which has a fast solution.

With a suitable fitness function, we can get a promising classifier. Both false negative rate and false positive rate can be controlled. Furthermore, since the GA is stochastic, we can extract distinct hypotheses along independent runs. Then, we can combine these hypotheses in a bagging-like mode, to form a bagged hypothesis, such as [8]:

$$BH(x) = Median\{h_i(x), i=1,\ldots,k\}$$

The key problem lies in the prior requirement: in classic GA, an individual is represented as a string over a finite alphabet. But most of the time, the number of observed events are huge, even infinite. For example, it's difficult for us to enumerate a person's actions in an office, because it's very likely he will do some unpredictable, though rational things. He may kneel down and pick up something, etc. In this case, the state space is too huge, so GA has difficulty to converge.

Another problem of GA is: it lacks a mechanism to deal with the uncertain nature of observations

### 3.5 Bayesian Networks

Bayesian networks (BN), also called belief networks or probabilistic networks, are graphical models to represent the interaction between variables visually [52]. A Bayesian network is composed of nodes and arcs between the nodes. Each node corresponds to a random variable, *X*, and has a value corresponding to the probability of the random variable, $P(X)$. If there is a directed arc from node *X* to node *Y*, this indicates that *X* has a direct influence on *Y*. This influence is specified by the conditional probability $P(Y|X)$. The network is a *directed acyclic graph* (*DAG*), i.e., there are no cycles. The nodes and the arcs between the nodes define the structure of the network, and the conditional probabilities are the parameters given the structure.

Bayesian networks have been used in various applications which initially were static, i.e., the nodes and links do not change over time. These applications involve determining the structure of the network; supplying the prior probabilities for root nodes and conditional probabilities for other nodes; adding or retracting evidence about nodes; and repeating the belief updating algorithm for each change in evidence.

Bayesian networks have become a popular representation for reasoning under uncertainty. Because of the uncertain nature of observations in activity recognition, Bayesian networks are very suitable for this domain. We will show how to use Bayesian networks in activity recognition by the Lumière project [27].

The Lumière project centers on harnessing probability and utility to provide assistance to computer users. They worked on Bayesian user models that can be employed to infer a user's needs by considering a user's background, actions, and queries. They tackled several problems: a) the construction of Bayesian models for reasoning about the time-varying goals of computer users from their observed actions and queries, b) gaining access to a stream of events from software applications, c) developing a language for transforming system events into observational variable. A derivative of Lumière project was embedded into Microsoft Office'97 as the

*Office Assistant*.

In this project, they constructed several models for different problem, including a BN to infer whether a user needs assistance, a BN to infer what a user's need is in Excel. They also tried to catch the temporal relationship by a dynamic BN in inferring what a user's need is. To speed the inference, they tried some strategies to simplifying the structures of models. Here, I will focus on the BN to infer whether a user needs assistance.

They constructed a small Bayesian network for this problem. The structure of this BN is shown in Figure 4. It represents the dependency between a pause after activity and the likelihood that a user would welcome assistance. According to the model, a user being in the state of welcoming assistance would shift the probability distribution of observing pauses in activity. The state of desiring assistance also influences the probability of detecting a recent search through multiple menus. They also considered the influence of a user's expertise and the difficulty of a task on the likelihood that a user will need assistance. We also note that a user will pause if he or she is distracted by events unrelated to the user's task. The difficulty of a task also directly influences the likelihood that a user will become distracted by other events.

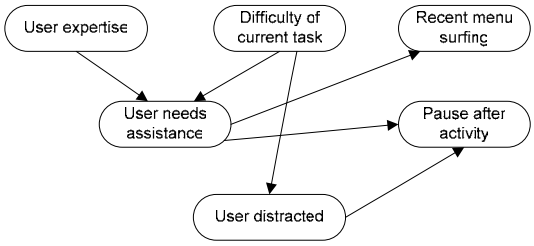



Figure 4: A Bayesian user model for inferring the likelihood that a user needs assistance, considering profile information as well as observations of recent activity.

They also created a special version of Excel that yielded information about subsets of mouse and keyboard actions, as well as information about the status of data structures in Excel files. The events included access to menus being visited and dialog boxes being opened and closed. In addition, they also gained information on the selection of specific objects, including drawing objects, charts, cells, row, and columns.

Then, they applied a language similar to what we discussed in Section2.2. These low-level, atomic events were transformed to the higher-level semantic events they employed in the above model. Based on these abstracted events, they did learning and inference.

Most of the time, the accuracy of Bayesian networks is promising. Activity modeling problems typically are dominated by uncertainty. Bayesian networks naturally model this nature and can encode rich expert knowledge.

Another advantage of this approach is that at run time, the expert can specify a cost matrix $C(\hat{y}, y)$ which specifies the cost of classifying the data as $\hat{y}$ when the true label of the data is $y$. This allows us to say, for example, that a false negative prediction is 10 times more serious than a false positive prediction [17]. With this utility information, the system can choose the classification that minimizes the expected cost:

$$\hat{y} = \arg\min_k \sum_y P(y \mid x) C(k, y)$$

Here, the sum over $y$ considers each of the possible classifications, computes the probability that this classification is true (according to $P(y|x)$), and then weights this by the cost of making decision $k$ when the true label is $y$. This can perfectly model our request to minimize *ECDA+ ECA*.

The problem is the inference speed of BN. As we have mentioned, exact inference in Bayesian networks is *NP-hard.* To overcome the speed problem, researchers have to simplify the structures of complex BN. Polytree structures will be the nice structures, which mean there is at most one undirected path between any two nodes in the network. They have a particularly nice property: the time and space complexity of exact inference in polytrees is linear in the size of the network.

Another problem is: BN is not good at modeling dynamic problems. For example, in the Lumière model, if the user surfs the menu twice, how can we reflect this in the model? We have to give up the early menu-surfing event. As we'll discuss, this problem will be mitigated by dynamic Bayesian networks.

### 3.6 Dynamic Bayesian Networks

Dynamic Bayesian networks (DBN) are an extension to standard Bayesian networks. First-order DBN work by maintaining two copies of a standard Bayesian Networks: one representing the beliefs at the current time $t$, and the other representing beliefs about the "next time" $t$+1. These two copies are referred to as time slices. A time slice in a DBN does not necessarily represent a fixed duration of time: rather, a transition between time slices occurs whenever a new piece of evidence arises. In the monitoring setting, this corresponds to the occurrence of an action, or the observation of such via the sensors [52].

To handle the potentially infinite number of parents, DBN make a *Markov assumption* – that is, the current state depends on only a finite history of previous states. The simplest one is the *first-order Markov process*, in which the current state depends only on the previous state and not on any earlier states. In other words, you need to make the future independent of past given the state. Using our notation, the corresponding conditional independence assertion states that, for all $t$,

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-1})$$

Hence, in a first-order Markov process, the laws describing how the state evolves over time are contained entirely within the conditional distribution $P(X_t|X_{t-1})$, which is called the *transition model* for first-order processes. The transition model for a second-order process is the conditional distribution $P(X_t|X_{t-1}, X_{t-2})$.

In addition to restricting the parents of the state variables $X_t$, we must restrict the parents of the evidence variables $E_t$. Typically, we will assume that the evidence variable at time $t$ depends only on the current state:

$$P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t)$$

The conditional distribution $P(E_t|X_t)$ is called the *observation model* (or, the sensor model). Notice the direction of the dependence: the "arrow" goes from state to observation values because the state of the world causes the sensors to take on particular values.

DBN include hidden Markov models and Kalman filters as special cases. Like BN, there are mature training and inference methods for DBN. We will show how to use dynamic Bayesian networks in activity recognition by an activity recognition system in MUD [3].

In this project, the domain is the "Shattered Worlds" Multi-User Dungeon (MUD), an adventure game which resembles the real world. It is a text-based virtual reality game where players compete for limited resources in an attempt to achieve various goals. The MUD has over 4700 locations, more than 7200 actions, and 20 different quests (goals). The objective of the project is to determine, as early as possible, which quest a player is attempting, and to predict which action a player will perform in the next move and which location a player will go to next.

To achieve the above object, the system must first learn which actions and positions (or, sequences of actions and positions) tend to lead to a particular quest. This information is obtained from previous instances of completed quests during a training phase and modeled by means of a DBN. During the inference phase, the DBN is used to predict a player's quest, next action and next location. To achieve this effect, every time a player performs an action, the system updates the probability that the player is trying to achieve each of the quests, perform each of the actions and move to each of the locations.

The dynamic BN is shown in Figure 5:

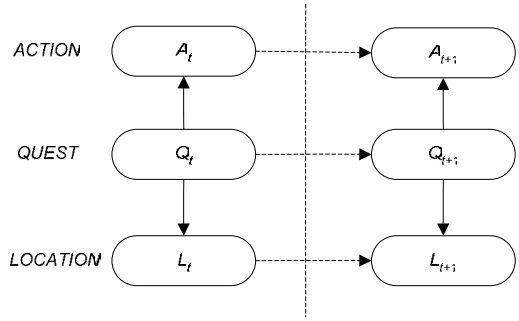



Figure 5: the dynamic Bayesian network for the MUD.

The domain variables are represented as nodes in the belief networks:

*Action* (*A*): This variable represents the possible actions a player may take in the MUD, which they took to be the first string of non-blank characters entered by a user, plus the special other action, which includes all previously unseen actions. For the results given in that paper, the state space size, $|A|$, is 7259.

*Location* (*L*): This variable represents the possible locations of a player, plus the special other location, which includes all previously unseen locations. For the results given in that paper, the state space size, $|L|$, is 4722.

*Quest* (*Q*): This variable represents the 22 different quests a player may undertake, including the other quest, which includes all previously unseen quests, and the null quest. The variable representing the previous quest achieved is set to null if the user has just started a session.

Actually, this network is not a pure dynamic Bayesian networks: the action and location may change over time, but it is assumed that a player's current quest does not change.

They collected 4981 runs. Among them, 80% are for training, 20% are for testing. The result show the system can correctly predict more than 80% quests.

Dynamic Bayesian networks can capture the temporal natures of activity recognition. The structures can be much simpler than Bayesian networks. Since they can naturally reflect the temporal correlation between events, the accuracy is promising. As Bayesian networks, they can also easily model our request to minimize *ECDA+ ECA*.

The problem is still the inference speed. The structure of the networks can be fairly simple, but the number of possible values of each node makes its training and inference a computationally complex task. It is proved that in almost all cases, the per-update time is exponential in the number of state variables.

Another problem lies in its representation power: because of their Markov assumptions, although DBN can model change in fluent values over time, they are inadequate for monitoring activities with quantitative temporal constraints [13]. For example, a DBN can't model the belief that lunch normally occurs 3-4 hours after breakfast. One conceivable way to incorporate such constraints would be to assign clock times and durations to each time slice and construct networks with many slices. However, assigning appropriate times and interval sizes to each time slice is very difficult. Large intervals result in a too coarse model, making it impossible to represent constraints that are smaller than the interval size; on the other hand, small intervals increase the size of the network and make inferences intractable.

In the following section, we will introduce some methods, trying to mitigating these problems.

### 3.7 Variations of Bayesian Models

A lot of variations were designed based on Bayesian networks. Generally speaking, there are two trends: one is trying to make models more complex in order to improve

the accuracy. Often, complex models can encode more expert knowledge. Some more complex models have been proposed, such as *Object Oriented Bayesian Networks*, *Probabilistic Relational Model, Quantitative Temporal Bayesian Networks*, *Conditional Random Fields*, *Averaged Perceptron*, *Hidden Markov SVMs*, *Max Margin Markov Nets*, etc. Another trend is trying to simplify models in order to speed the inference, since most real applications are on-line and need fast inference. The ordinary methods are a) remove dependent relationship, and b) cluster similar variables nodes into one node. In this way, they may get a simpler structure, even a polytree sometimes. This approach depends on the knowledge of experts and lacks of special theories.

There have been too many discussions about OOBN, PRM and discriminative methods. Instead of this, I'd like to introduce some efforts about how to encode quantitative temporal constraints in DBN, and how to simplify Bayesian models.

Colbry proposed *Quantitative Temporal Bayesian Networks* (*QTBN*), intending to combine the benefit of both BN and DBN [13]. This model includes two components: an auxiliary BN and a main DBN.

The BN is used to encode the likelihood of events occurring during particular intervals. It is automatically constructed by building a single node for every action in the plan and an arc between the nodes for every Temporal Constraint associated with the plan. One additional node, the TimeReferencePoint, indicates an arbitrary starting time to ground quantitative temporal references. The time intervals are sized based on the granularity that the action requires. The values assigned to each node represent a belief distribution over time. From these belief distributions, we can extract information about what has happened in the past and predict what will happen in the future. We can also extract information about the current time slice for use in the DBN component.

At any time, the information in the DBN represents and reasons about a small interval of time, called the CurrentTimeInterval. The DBN retrieves information about the CurrentTimeInterval from the BN and then operates like a standard DBN. In the DBN, there is a special node influencing the action, which is called the Temporal Influence Node ($A_{TIN}$). $A_{TIN}$ summarizes all of the quantitative temporal beliefs about action $A$ for the CurrentTimeInterval.

Figure 6 shows an example of QTBN: how to withdraw money from ATM. Usually, you will input number about 3 seconds after you slide the card, and you will take the money about 15 seconds after you input the number.

Node *FinishedActions* is a persistent variable. It contains a hash table, recording our cumulative belief that each action has occurred or is occurring. The *Current Action* is influenced by *FinishedActions* and *ActionTIN,* and influences the *hand position*.

As stated earlier, a TIN can be considered a summary of all temporal information related to the action it influences and to the CurrentTimeInterval. A TIN contains a hash table whose entries are each action. And the values of the hash table cell are determined by querying the BN. The result of the query, "What is the probability that action $A_i$ will occur in the CurrentTimeInterval?" is set to the $A_i$ cell in the hash table of the TIN. This query is performed by probing the associated action node in the BN and extracting the portion of the returned probability distribution that corresponds to the current time interval.

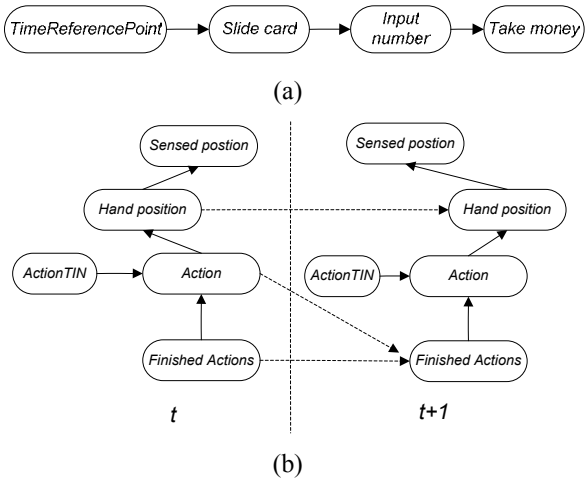



Figure 6: a Quantitative Temporal Bayesian Network. (a) is the BN component to represent the temporal dependencies of actions; (b) is the DBN component.

Also, the BN/DBN Interface provides an information channel between the DBN and BN through the various action nodes. This interface has two functions: a) UpdatePredictions, which extracts the summary temporal data from the BN to be used to influence reasoning in the DBN. b) RecordHistory, which extracts data from the DBN for use as historical data in the BN; this will update the BN's CPTs to reflect beliefs about what has occurred since the last time change.

This is an interesting attempt to combine the benefit of BN and DBN, though they didn't report accuracies in the paper. For the detail, please refer to [13].

Meanwhile, there have been some works on represent BN in compact way, such as decision trees and decision graph [26, 27].

In an autominder project [26], the task was to infer whether the desktop user is interruptible according to windows events, visual & acoustical analyses, and online calendars. This needs fast inference, so they tried Bayesian networks instead of dynamic Bayesian networks. But this is not enough: since the only unobserved and interested variable is *Interruptible*, they used a decision graph representing the compact encoding of the probability distribution underlying the *Interruptible* variable of the Bayesian network. Part of the decision graph is shown in Figure 7.

The bar graphs at the leaves of the tree represent probability distributions over high, medium, and low *costs of interruptability* (ordered, top to bottom, from high to low) for sets of observations represented by the paths leading to the leaves. The paths to the leaves identify important combinations of events to decide the probability distribution over *costs of interruptability*. The paths branch on key observations drawn from the user's calendar and from the real-time activity event stream, including patterns of presence, application usage, and perceptual events.

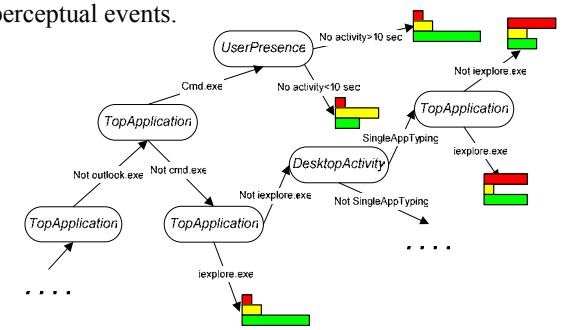



Figure 7: part of decision graph encoding probability distribution for the *Interruptible* variable.

In this way, a Bayesian network is transferred into a decision tree. Thus, it speeds the inference greatly.

In Lumière project [27], Horvitz et al tried another way to speed inferences: modeling a temporal problem as a naïve Bayesian model. The task is: they tried to infer the user's goal according to the user's actions. They approximated this temporal problem based on direct assessment of parameters of functions that specify the probabilities of observations conditioned on goals. Inputs of the function are the amount of time that has transpired between the observation and the present moment. The intuition behind the approach is that observations seen at increasingly earlier times in the past have decreasing relevance to the current goals of the user.

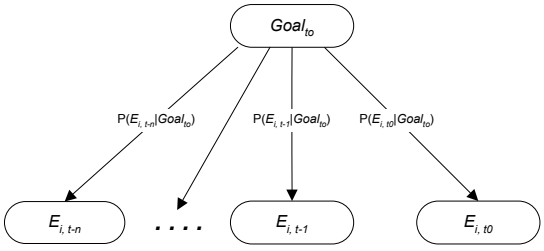



Figure 8: Formulation of the temporal reasoning problem as naïve Bayesian model. They directly assess conditional probabilities of actions as a function of the time that has passed since actions occurred.

$E_{i,t}$ is the observation of variable $i$ at time $t$. They made the assumption that the rates of likelihood are independent of one another, conditioned on goals.

This is essentially a naïve Bayesian model, which can guarantee the inference speed.

# 4 Possible Improvement for Activity Recognition

In real applications, people prefer to the off-the-shelf learning methods, because a) most applications should be developed in definite time, off-the-shelf methods can guarantee the application is completed in time; b) the performance of a novel method is usually unpredictable, the developer tend to make their products more stable. Thus, they seldom try new ideas, unless there is a very urgent need for accuracy.

In this section, I'd like to explore some feasible ideas. There are two goals here: one is to make the development process easier, the other is to make the prediction more accurate.

## 4.1 Applying Expert Knowledge

State-of-the-art methods for constructing knowledge based systems can capture the domain knowledge, but they can't learn from training data. Furthermore, knowledge engineering is labor intensive and not well suited for some of the kinds of knowledge needed for effective performance. State-of-the-art methods for machine learning can learn from training data, but they provide no way to exploit domain knowledge. Furthermore, they generally require very large amounts of training data in order to achieve high performance levels [18]. If we can combine more expert knowledge into learning system, it may reduce the needed number of training data.

Currently, the main way to make use of expert knowledge is for the data analyst to study the domain knowledge and design a special-purpose learning system that incorporates this knowledge [18]. In probabilistic model problem, this is mainly done by deciding the structures of probabilistic models (especially BN). Though we can learn the structures of probabilistic models from the training data, the result is generally worse than those with handcraft structures. I'll discuss some methods to learn the structures of probabilistic models from knowledge, instead of data. In this way, we mimic the process that experts construct the structures of probabilistic models by hand.

Huber proposed a method to construct Bayesian networks from descriptions of plans [31]. These descriptions are organized into *Knowledge Areas*, or *KA*s. KAs specify how plans are selected given the current goal (its *purpose*) and situation (its *context*). KAs also specify a procedure, called the KA *body*, which it follows while attempting to accomplish its intended goal. Experts use KA to describe how plans are executed. Then based on these descriptions, Bayesian networks are constructed automatically.

Basically, the construction process can be summarized as the following:

- Create a variable representing the goal to be achieved by the KA.

- Create a new variable for each action in the KA. Each of these new variables depends on the goal variable.
- To model the temporal relationship, create an arc between sequential-happening actions
- Add observation variables, associate them with the corresponding actions.
- Add the *context* variables, which depends on the goal variable
- Using an "*inhibitory link*" mechanism to process multiple goals, multiple plans

A simple example is given in Figure 9:

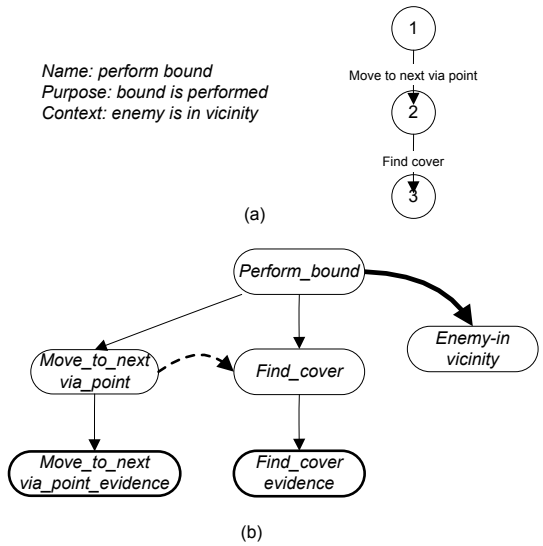



Figure 9: (a) is a simple KA; (b) is the corresponding Bayesian network.

This KA says that in order to achieve the goal of accomplishing a "bound" goal, the operations of moving to the next location (the *via point*) and finding a place of concealment must be accomplished. Knowing this, if an observer were to see an agent moving toward a grove of trees, the observer might predict that the observed agent was about to enter the grove. As we can see, the constructed Bayesian network is quite reasonable: it reflects our thought.

Intel Research developed a toolkit called *Probabilistic Activity Toolkit* (*PROACT*) [47]. In this project, they tried to infer the user's activity from the objects involved in the activity. Two interesting tings were explored: a) convert natural-language-like knowledge into probabilistic models, and b) mine probabilities from Internet. In this approach, they didn't need any training data at all.

PROACT's activity model is restricted to linear sequences of sub-activities which are annotated object information. Figure 10 gives an example of how to model the activity "making tea." The activity consists of three consecutive sub-activities, drawn as circles, including a) boiling water, b) steeping, and c) flavoring the tea.

The dotted arrow denotes the probability that an object is involved in an activity. In Figure 10, for instance, we expect to see a kettle 70% of the time that we are boiling water, whereas sugar is involved in the mixing phase 40% of the time.

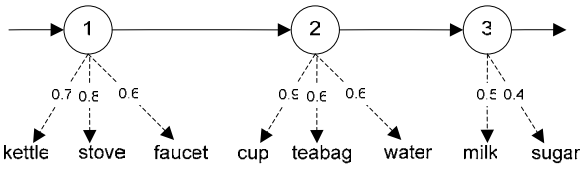



Figure 10: PROACT Model for Making Tea.

To model activities more easily, PROACT allows activities to be specified as text documents that are structurally very similar to recipes. Figure 11 provides an example. Each document has a title, an optional list of objects involved, and a step-by-step description of how to perform the activity. The mining engine then converts the document into an activity model by interpreting the steps as sub-activities and the objects mentioned in each step as the set of objects involved. They identified the objects mentioned in each step (highlighted in bold in the figure) using natural language processing technology.

Making Tea:

1. Fill a **kettle** from the **faucet**. Place kettle on the **stove** and boil.
2. Pour hot **water** into a **cup**, filling ¾ of the cup. Immerse **teabag** in cup for two minutes and dispose of teabag.
3. Add **milk** and **sugar** to taste.

Figure 11: Steps for Making Tea.

PROACT determines the object involvement probabilities $p$ in a novel manner. The method relies on a "Mirror Assumption": if the name of an activity $a_n$ co-occurs with the name of an object $o_n$ in human discourse, then activity $a_n$ is likely to involve object $o_n$ in the physical world. They postulated on this basis that if $a_n$ occurs on $n_1$ pages on the web (which they treated as a compendium of human discourse), and there are $n_2$ pages containing both $a_n$ and $o_n$, then $Pr(o_n | a_n) \approx n_2 / n_1$. They obtain these numbers via the *Google programming API*.

They tested the toolkit in Activity of Daily Living. The overall precision rate is 88% and recall rate is 73%, which is promising.

### 4.2 Applying Hierarchical Information of Events

As we've mentioned, events (the observations of sub-activities) can be fitted into a hierarchy. Applying this hierarchical information may improve the performance of activity recognition in three ways: a) reduce the needed

number of training data, b) reduce the inference time, and c) improve the inference accuracy.

Many important military and civilian problems have the property that there is relatively little data available. For example, considering the development of cognitive assistants for human-computer systems, normally it will require the human user to provide many hours of labeled training data to the learning system [18]. If we can reduce the required data, we can make such application more pratical.

In all forms of learning, the primary way to reduce the need for training data is to incorporate some kind of prior knowledge [16], such as the hierarchical information.

Several kinds of methods have been explored. The naïve method is to abstract atomic events to some high-level events, and then make learning and inference based on these high-level events. We can use the activity language mentioned in Section2.2 to do this job. Actually, this is what the Lumière project did. In the Lumière project [27], the hierarchy also did some disambiguation things: obviously it is useful to include an efficient means for abstracting sets of low-events into event classed specified as disjunctions of events (e.g., they specified that a user saving a file via a toolbar icon or keyboard action is to be generalized to *file saved*). A carefully designed hierarchy can do this job.

Another method is called *shrinkage* [40]. The essential idea is to "borrow" information from its neighbors. Shrinkage smoothes parameter estimates of a data-sparse child with its parent in order to obtain more robust parameter estimates. It is a statistical technique to reduce the variance of an estimate by averaging it with estimates for larger populations that include the target one.

We can think shrinkage as a mixture model:

$$P(x) = \sum_{\alpha \in A(x)} \lambda_\alpha P(\alpha)$$

where $A(x)$ represents the set of $x$'s ancestors. Also

$$\sum \lambda_\alpha = 1 \quad \& \quad \lambda_\alpha \geq 0$$

Shrinkage is especially efficient when hierarchy is large or training data for each class is sparse. It has been proven that shrinkage improves parameter estimation and reduces classification error greatly. McCallum also showed that a class hierarchy can be used to exponentially reduce the amount of computation required to classify documents without sacrificing significant classification accuracy [40].

Anderson applied shrinkage to adapt web navigation [1], which is a very similar task to activity recognition. The main goal here is to predict which webpage the user wants to browse next.

Some webpages are very similar, and will link to other similar webpages. So they applied shrinkage here, namely *Relation Markov Model* (*RMM*).

RMM groups pages of the same type into relations, with each relation described by its own set of variables. For example, in Figure 12, one relation might be "product description page", with a variable "product" representing the product the page describes, and "stock_level" representing whether the product is in stock. Additionally, these variables themselves are grouped together, forming a hierarchy of values.

Provided with a hierarchy, RMM does learning and inference using shrinkage method. RMM can make useful prediction possible in very large state spaces, where many (or most) of the states are never observed in the training data.

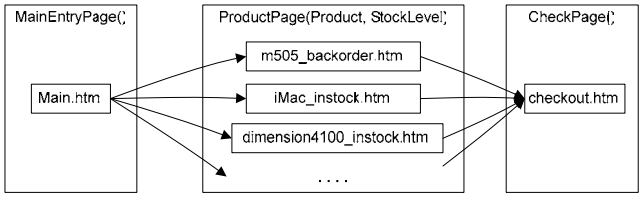



Figure 12: State abstraction for the relational Markov model. The abstractions are depicted as square boxes, labeled with their relations and arguments, and surrounding their ground states.

### 4.3 Applying State-of-the-art Learning Models

Some more complex and more accurate models have been proposed for sequential problems, such as *Conditional Random Fields* [38], *Averaged Perceptrons* [14], *Hidden Markov SVMs* [6], *Max Margin Markov Nets* [54], etc. These models should be able to improve the accuracy of activity recognition, since activity recognition has a temporal nature.

These models have been tried in several domains, such as Optical Character Recognition, Name Entity Recognition, Protein Secondary Structure Prediction, etc. In most of these applications, the training set is small (thousand-magnitude), and the computation time is not critical.

Until now, none of these models have been tried in activity recognition. Actually, these models have all the preferred characteristics according to our criteria set: the accuracies are promising, and the inference speeds are fast. This time, the problem will be training speed because the training processes of all these models are time-consuming. Compared with them, even the training of HMM becomes very "efficient"!

As we can imagine, most of the activity recognition applications should update their inference engine frequently. There are two reasons: a) the users of these applications may change, e.g., the may sell their cognitive assistants to other people; b) the users' habits may change from time to time. Thus we need retrain the inference engine after we accumulate the data for some time. This can be done by incremental learning or total renewal of the engine. The accumulated data will be huge. Thus, if the training time is too long, saying more than 12 hours, it will affect our use of the applications.

We can try these state-of-the-art models in some general situations, which don't need retraining according to

different users and different periods. When faster training methods and faster computers are developed, we can apply these models in more specific problems.

## 5 Summary and Conclusions

In this paper, I made a survey of existing methods for activity recognition. Activity recognition observes a sequence of measurements, tries to guess the status of the subjects, and responds to it accurately and timely. Depending on its goal, it can be classified into activity monitoring or activity prediction.

I showed we can describe activities with a hierarchy. Such a hierarchy can help me in many aspects. For example, we can use these hierarchies to do feature designs by abstracting atomic events into high-level events. To facilitate feature designs, I synthesized a language to describe activities. This language can also be used to do Genetic Algorithms.

To evaluate different activity recognition methods, I used two criteria. One is the inference time, since most activity recognition is on-line. Another is the sum of the expected cost of delayed action and the expected cost of annoyance. An optimal recognition method should minimize these two values.

I discussed the special methods of data preprocessing in activity recognition. Then, I summarized 5 off-the-shelf learning methods in activity recognition: decisions trees, neural networks, generic algorithms, Bayesian networks and dynamic Bayesian networks. Among them, probabilistic models are the most popular, so I discussed some variations of probabilistic models furthermore.

Finally, I discussed some methods that possibly improve the performance of activity recognition. There are two goals: one is to make the development process easier, and the other is to make the prediction more accurate. I investigated how to construct probabilistic models automatically from expert knowledge, and how to apply hierarchical information in learning. I also discussed the possibility that we apply state-of-the-art learning models in activity recognition.

**References**


[1] C.R. Anderson, P. Domingos, D.S. Weld. Relational Markov Models and their application to adaptive web navigation. In *SIGKDD02*, 2002.

[2] C. R. Anderson and E. Horvitz. Web Montage: A Dynamic Personalized Start Page. The *11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002

[3] D. Albrecht, I. Zukerman, et al. Towards a Bayesian model for keyhole plan recognition in large domains. In *Proc. Of the 6th International Conference on User-Modeling*, 1997.

[4] J. Allan, R. Papka and V. Lavrenko. Online new event detection and tracking. In *Proc of 21st ACM-SIGIR*, 1998.

[5] E. Alpaydin. *Introduction to Machine Learning*. To be published by MIT Press, 2004.

[6] Y. Altun, I. Tsochantaridis and T. Hofmann. Hidden Markov Support Vector Machines. In *Proc. of ICML2003*, 2003.

[7] S. Andrews, L. Cai, et al. Astrology: the study of Astro Teller. *ICML04 Workshop "Physiological Data Modeling - A Competition",* 2004.

[8] J. Azé, N. Lucas and M. Sebag. PDMC: a genetic ROC-based classifier. *ICML04 Workshop "Physiological Data Modeling - A Competition",* 2004.

[9] J. Boger, G. Fernie, P. Poupart and A. Mihailidis. Using a POMDP controller to guide persons with dementia through Activities of Daily Living. In *Proc of UbiComp03*, Seattle, WA, 2003.

[10] E. Camp and M.Chan. Detecting Abnormal Behavior by Real-tiem Monitoring of Patients. *AAAI workshop on "Automation as Caregiver"*, 2002.

[11] C. Campbell and K. Bennett. A linear programming approach to novelty detection. In *Proc. of NIPS2001*, 2001.

[12] M. Chan, C. Hariton, P. Ringeard and E. Campo. Smart house automation system for the elderly and the disable. In *IEEE international Conference on Systems, Man and Cybernetics*, pages 1586-1589, 1995.

[13] D. Colbry, B. Peintner and M. Pollack. Execution monitoring with Quantitative Temporal Bayesian Networks. In 6th *International Conference on AI Planning and Scheduling*, April 2002.

[14] M. Collins. Discriminative training methods for Hidden Markov Models: theory and experiments with perceptron algorithms. In *Proc. of EMNLP02*, 2002

[15] N. Cristianini and J. Shawe-Taylor. *Support vector machines and other kernel-based learning methods*. Cambridge Press, 2000.

[16] T.G. Dietterich, D. Busquets, R.L. Mantaras, C. Sierra. Action Refinement in Reinforcement Learning by Probability Smoothing. In *Proc of ICML2002*, 2002.

[17] T.G. Dietterich. *Learning and Reasoning*. Technical report, School of Electrical Engineering and Computer Science, Oregon State University, 2003.

[18] T.G. Dietterich. *Proposal for KI-LEARN project*. School of Electrical Engineering and Computer Science, Oregon State University, 2003.

[19] T.G. Dietterich, A. Ashenfelter and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. of ICML2004*, 2004.

[20] J. Eisenstein and A. Puerta. Adaptation in automated User-Interface design. In IUI2002, 2002.

[21] T. Fawcett and F. Provost. Activity monitoring:

notice interesting changes in behavior. In *KDD99*, 1999.
[22] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Relational Data Mining*. Springer-Verlag, 2001.
[23] A. Guralnik and K. Haigh. Learning models of human behaviour with sequential patterns. *AAAI workshop on "Automation as Caregiver"*, 2002.
[24] K. Haigh and H.A. Yanco. Automation as caregiver: a survey of issues and technologies. *AAAI workshop on "Automation as Caregiver"*, 2002.
[25] D. Heckerman and E. Horvitz. Inferring informational goals from free-text queries: a Bayesian approach. In *Proc UAI98*, Morgan Kaufmann, 1998
[26] E. Horvitz and J. Apacible. Learning and reasoning about interruption. In *Proc. of ICMI03*, 2003.
[27] E. Horvitz, J. Breese, D. Heckerman, D. Hovel and K. Rommelse. The Lumière Project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. UAI98*, 1998
[28] E. Horvitz, A. Jacobs, D Hovel. Attention-Sensitive Alerting. In *Proc. UAI99*, 1999.
[29] E. Horvitz, C.M. Kadie, T. Paek, D. Hovel. Models of Attention in Computing and Communications: From Principles to Applications. *Communications of the ACM* 46(3):52-59, March 2003.
[30] E. Horvitz, P. Koch, et al. Coordinate: Probabilistic Forecasting of Presence and Availability. In *Proc. of UAI02*, 2002.
[31] M.J. Huber, E.H. Durfee and M.P. Wellman. The automated mapping of plans for plan recognition. In *Proc. of UAI94*, 1994.
[32] S.E. Hudson, J. Fogarty, et al. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *Proc. of CHI2003*, 2003.
[33] A. Jameson. Numerical uncertainty management in user and student modeling: an overview of systems and issues. *User Modeling and User-Adapted interaction*, 5:193-251, 1996.
[34] L.P. Kaelbling, M.L. Littman and A.R. Cassandra. Planning and action in partially observable stochastic domains. *AI Magazine*, 101:99-134, 1998.
[35] H. Kautz, O. Etzioni, D. Fox and D. Weld. *Foundations of Assisted Cognition Systems*. UW CSE Technical Report, March 2003
[36] E. Keogh, S. Lonardi and B. Chiu. Finding surprising patterns in a time series database in line time and space. *In SIGKDD02*, 2002.
[37] K. Koile, K. Tollmar, et al. Activity Zones for Context-Aware Computing. In *UbiComp2003*, 2003.
[38] J. Laferty, A. McCallum and F. Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML2001*, 2001.
[39] N. Lesh and O. Etzioni. A sound and fast goal recognizer. In *Proc. of IJCAI95*, 1704-1710, 1995.
[40] A. McCallum, R. Rosenfeld, T. Mitchell and A.Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. *In Proc. of ICML98*, 1998.
[41] A. Mihailidis and G.R. Fernie. Context-aware assistive devices for older adults with dementia. *Gerontechnology*, 2(2):173-189, 2002.
[42] A. Mihailidis, G.R. Fernie and J.C. Barbenel. The use of artificial intelligence in the design of an intelligent cognitive orthosis for people with dementia. *Assistive Technology*, 13: 23-39, 2001.
[43] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
[44] N. Oliver and E. Horvitz. Selective Perception Policies for Guiding Sensing and Computation in Multimodal Systems: A Comparative Analysis. *Proceedings of the 5th International Conference on Multimodal Interfaces*, 2003.
[45] D.J. Patterson, L. Liao, D. Fox and H. Kautz. Inferring high-level behavior form low-level sensors. In *Proc of UbiComp03*, Seattle, WA, 2003.
[46] J. Petzold, F. Bagci, W. Trumler, and T. Ungerer. Global and Local State Context Prediction. *Artificial Intelligence in Mobile System 2003*, in conjunction with *Ubicomp2003*, 2003.
[47] M. Philipos, K.P. Fishkin, et al. *The probabilistic activity toolkit: towards enabling activity-aware computer interfaces*. Intel Research Seattle Technical Memo IRS-TR-03-013, December 2003.
[48] W.Pohl. Learning about the user – user modeling and machine learning. *ICML96 Workshop "Machine Learning meets Human-computer Interaction"*, 1996.
[49] M.E. Pollack, C.E. McCarthy, et al. Autominder: a planning, monitoring and reminding assistive agent. In 7th *International Conference on Intelligent autonomous Systems*, 2002.
[50] M. Pollack. *Plans as Complex Mental Attitudes.* MIT Press, 1990.
[51] D.V. Pynadath and M.P. Wellmann. Accounting for context in plan recognition, with application to traffic monitoring. In *Proc. of UAI95*, 1995.
[52] S. Russell and P. Norvig. *Artificial intelligence: a modern approach* (2nd version). Prentice Hall, 2002.
[53] J. Ruvini and C. Dony. Learning users habits: the APE project. *AAAI workshop on "Automation as Caregiver"*, 2002.
[54] B. Taskar, C. Guestrin and D. Koller. Max-margin Markov networks. In *Proc. of NIPS2003*, 2003.
[55] Ubiquitous Computing. http://www.ubiq.com/ hypertext / weiser/UbiHome.html
[56] G. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *KDD98*, 1998.