Predicting user behavior using transition probability

Master Thesis

Author
Rolf Heukels

Supervisor
dr. ir. Nirvana Meratnia

Graduation Committee
dr. ir. Nirvana Meratnia
Prof.dr. Paul J.M. Havinga
ir. Hans Scholten

Enschede
Faculty of Electrical Engineering, Mathematics and Computer Science
Pervasive Systems research group

Abstract

As energy becomes more expensive and the environmental effects of fossil fuels become more apparent, a greater interest arises in reducing our energy needs. Projects [1][2][3][4][5]intended to find new ways of making our everyday lives more energy efficient have become an integral part of the struggle to sustain our current quality of living. These projects target not only large, industrial processes, but also processes on a domestic scale which have a more direct approach to changing the way we consume energy.

The Go-Green [6]project is one of these projects focusing on making our homes a “greener” place. Inhabitants are monitored and information is acquired about the usage of appliances, the behaviour of people, and the state of the surroundings. Energy is saved by learning user habits and reacting to inefficient energy consumption. Inefficiencies can take on the form of appliances that are left running unintentionally, or a central heating system that is warming up an empty house. However, dealing with wasting energy and controlling the way people use their home, should at no point compromise their living comfort. In fact, user comfort is to be increased by applying the same knowledge of user habits to automatize the interaction with devices.

Learning and predicting user habits require a suitable machine learning technique. Research has shown that there are several candidates, all with their own advantages and disadvantages. The specific requirements for this project made it difficult to find a single technique that is suited for our needs. Especially the behavior prediction aspect seemed insuperable for techniques that are primarily applied for behavior classification.

In this thesis we design a new machine learning technique, specifically aimed at predicting user behavior. “Transition probability”, as this technique is called, is explained and an overview is given about its origin and its possible future. We discuss the theory behind this technique and provide a practical validation of our theory by means of simulation. Several key aspects are evaluated to obtain a clear view of performance and applicability to our problem.

Table of Contents

Chapter1 Introduction....10
Section1.1 Problem statement....10
Section1.2 Thesis overview....10
Chapter2 Related work....11
Section2.1.1 Decision trees....11
Section2.1.2 Linear machines....12
Section2.1.3 Association rules....14
Chapter3 Transition probability....16
Section3.1 Inspiration....16
Section3.1.1 Nearest neighbor classification....16
Section3.1.2 Markov chains....17
Section3.2 Model of the environment....18
Section3.2.1 States....18
Section3.2.2 Transitions....19
Section3.2.3 Sectors....19
Section3.3 Prediction algorithm....20
Section3.3.1 Initialization....20
Section3.3.2 Influence distribution....21
Section3.3.3 Final processing....23
Section3.3.4 Pseudocode....24
Section3.4 Relevance distribution over time....26
Chapter4 Implementation....27
Section4.1 Architecture....27
Section4.2 Functionality....29
Section4.3 Graphical user interface....30
Section4.4 Algorithm implementation and optimization....32
Chapter5 Evaluation....34
Section5.1 Data set....34
Section5.2 Performance metrics....36
Section5.2.1 Prediction accuracy....36
Section5.2.2 Latency....36
Section5.2.3 Memory usage....36
Section5.3 Impact analysis of changing conditions....37
Section5.3.1 Amount of training data....37
Section5.3.2 Level of variance in behavior....39
Section5.3.3 Different types of activities....41
Section5.3.4 Different algorithm parameters....42
Section5.4 Impact analysis of sub-techniques....44
Section5.5 Comparison with other techniques....45
Section5.5.1 Accuracy comparison....45
Section5.5.2 Accuracy function comparison....47
Section5.5.3 Computation time and memory usage....48
Section5.5.4 Conclusion....48
Chapter6 Conclusion....49
Section6.1 Future work....50
Section6.1.1 Relevance distribution over time....50
Section6.1.2 Data representation....50
Section6.1.3 State duration....50

List of figures

Figure 2.1.1.1: An example of a decision tree....12
Figure 2.1.2.1: An example of a linear machine....13
Figure 3.1.2.1: An example of a Markov chain....17
Figure 3.2.1.1: An example of multiple states and the transitions between them....18
Figure 3.3.1.1: An example of preselection of transitions within range....20
Figure 3.3.2.1: Simple point reassignment....21
Figure 3.3.2.2: Recursive point reassignment....21
Figure 3.3.2.3: A loop....22
Figure 3.3.3.1: A point processing example....23
Figure 4.1.1: System architecture....27
Figure 4.3.1: Output window....30
Figure 4.3.2: Debugging window....31
Figure 4.4.1: Linear relevance distribution....32
Figure 4.4.2 : Distribution based on transitions....32
Figure 4.4.3: Distribution based on bundles....33
Figure 5.3.1.1: Detection rate....37
Figure 5.3.1.2: False alarm....37
Figure 5.3.1.3: Average latency....38
Figure 5.3.1.4: Memory usage....38
Figure 5.3.2.1: Detection rates....39
Figure 5.3.2.2: False alarms....40
Figure 5.3.2.3: Latency....40
Figure 5.3.3.1: Detection rates....41
Figure 5.3.3.2: False alarms....41
Figure 5.3.4.1: Detection rates....42
Figure 5.3.4.2: False alarms....42
Figure 5.3.4.3: Detection rates....43
Figure 5.3.4.4: False alarms....43
Figure 5.3.4.5: Detection rates....43
Figure 5.3.4.6: False alarms....43
Figure 5.4.1: Detection rates....44
Figure 5.4.3: False alarms....44
Figure 5.4.3: Average latencies....44
Figure 5.4.4: Memory usages....44
Figure 5.5.1.1: Time slice accuracies for HMM,CRF and TP....46
Figure 5.4.2.1: Optimal prediction accuracy....47
Figure 5.4.2.2: Optimal classification accuracy....47

Chapter1 Introduction

Section1.1 Problem statement

The Go-Green project [6] involves a group of people living in a house. Sensors placed in the house, monitor these people and their power consumption. Each appliance has its own sensor, recording when the appliance is being used and how much power that consumes. Sound is recorded in order to get more information about who is living in the house and what activity is performed. This in combination with motion sensors, makes it is possible to also keep track of where everyone is. Room temperature and weather conditions are monitored to provide more of a general context.

In the spirit of energy saving, this project aims to reduce power consumption in a common household. Not only by keeping track of energy usages, but also by actively assisting the user in making energy saving decisions. Appliances may be turned of automatically when they are no longer in use or left on unintentionally. These decisions, however, should at no point compromise user comfort. It would be unacceptable if lights are turned off in a crowded room and people are literally left in the dark. If anything, user comfort should be increased by adapting to user behavior as much as possible. These seemingly conflicting interests must be resolved within some kind of compromise or trade-off.

Achieving these goals will be a matter of learning user habits. A learning technique is needed to make sense of all the different sensor inputs and to extract certain patterns in user behavior. If these patterns are successfully extracted, it is possible to make reasonable predictions about future behavior and to make useful decisions accordingly.

Section1.2 Thesis overview

In this thesis we design a new machine learning technique, specifically aimed at predicting user behavior. Chapter 2 describes a selection of existing learning techniques, each evaluated on their applicability to our problem, and each contributing to the design goals of our technique. Our new technique, called “transition probability”, is presented in chapter 3. The implementation details and the functionality of our system is described in chapter 4. Chapter 5 focuses on the performance of our algorithm. Performance figures on prediction accuracy, latency and memory usage are gathered, evaluated and compared to those of other techniques. Chapter 6 concludes the thesis and gives an overview on future work.

Chapter2 Related work

Before selecting transition probability as our solution, several other learning techniques [7][8][9] [10][11][12][13][14]were researched and evaluated. The following sections contain a selection of these techniques and describe how these techniques could be mapped to our problem.

Section2.1.1 Decision trees

Concept

Like any other tree, a decision tree has nodes, branches and leafs. A node represents a single test that evaluates a specific part of the input data to a finite number of possible outcomes. This decision may take several input features into account to come to a result. Each outcome is equivalent to a branch that leads to another node. Nodes no longer branch out at the bottom of the tree. These nodes are called leafs and they represent a final answer like “true” or “false” but they may also be classes or categories in the case of a classification problem. These categories however need to be mutually exclusive since only a single leaf will be selected as output. To come to this output the input follows a path, from the root of the tree down to one of leafs, that is dictated by the filtering properties of the tree.

Training

Training a decision tree is a matter of deciding what the tests are going to look like and what the order of evaluation is going to be. The goal is to make the tree as small and effective as possible.
A test will function in a certain way based on the kind of input features it evaluates. If the features are binary, the test will simply check whether it is true or false. However, a feature may also be numerical. In this case may involve checking if the value is within an interval. Deciding what this interval should be is done in the same way as finding the best order of evaluation.

Uncertainty reduction is the process of repeatedly finding the features that results in the highest information gain. Each test divides the samples in several partitions. The entropy of the partition is a measure that defines the amount of equality between the samples in this partition. If the partition only contains samples of the same category, the entropy is zero. Conversely the entropy will be one if the partition is very impure and each sample belongs to a different category. The information gain is equal to the amount of purification in the remaining partitions after each test.

Mapping

Putting the sensor data into a tree will result in binary nodes. These nodes have only two branches that correspond to the device being on or off. The date and time input consist of numeric values so nodes that test on these inputs will be more complex. A tree that will detect when a person is watching television or is likely to do so in the near future, may look like this:

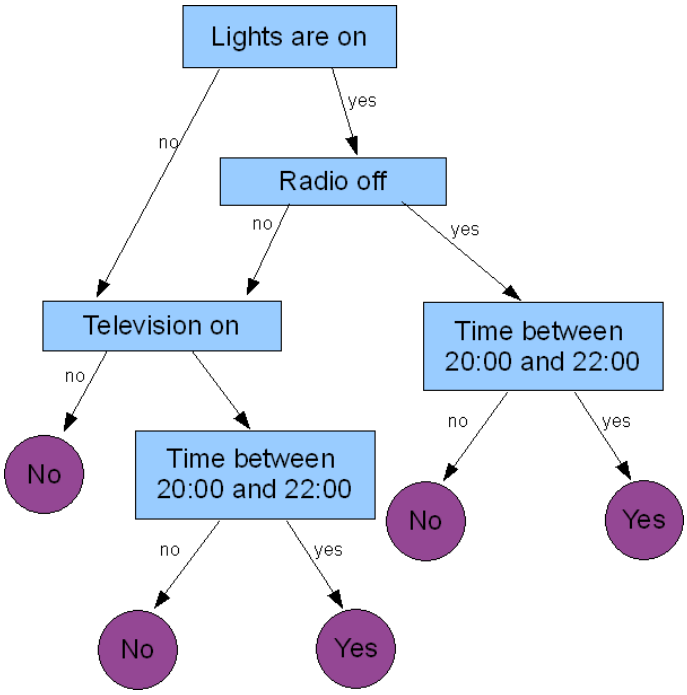


Figure 2.1.1.1: An example of a decision tree

Section2.1.2 Linear machines

Concept

The most basic neural network is that of a single neuron. This neuron will evaluate all inputs and return a binary result by comparing the weighted sum of the inputs with a threshold value. Such a configuration may be used for a single “true” or “false” decision. Commonly, decision problems involve several of these small tests to come to a final answer. This is when multiple neurons are connected to form a network.

One straightforward way of using neural networks for classification is by using a linear machine. This type of networks allows for classification into multiple categories. Neurons are arranged in parallel, each representing one category. Every neuron evaluates all inputs to calculate the weighted sum associated with its category. The neuron that produces the highest output dictates the final outcome in this “winner-takes-all” strategy.

Training

The goal of training is to find the correct weights associated with each input at each neuron. Correctness is defined by the total squared error over all training patterns. The training process iteratively tries to reduce this error by repeatedly applying two rules; If the machine classifies the pattern correctly, no changes are made to any of the weights. If a category *u* pattern is mistakenly classified in category *v*, then the weights belonging to the *u* neuron are increased and the *v* weights are decreased by some constant. These weight changes only apply to the weights at the inputs defined in the pattern in question.

Mapping

For every class there needs to be a neuron representing it. In this example, input is to be classified in one of two classes, e.g. listening to the radio and watching television. The network has been trained with a training set that describes the following habits:

Condition	Action
Morning, radio off, light off, television on	Television
Morning, radio on, light on, television off	Radio
Evening, radio off, light on, television on	Television

A simplified version of this linear machine is depicted bellow. For the sake of clarity, this version features only a small selection of inputs and the weights are either 1 or 0.

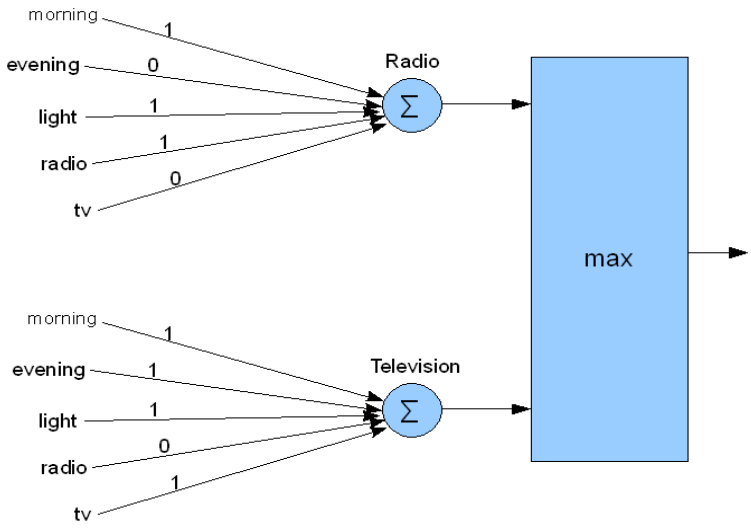


Figure 2.1.2.1: An example of a linear machine

Section2.1.3 Association rules

Concept

This form of data mining is used to find useful relations between features in a training set. These relations are expressed in the form of simple rules that have some features on the left side implying features on the right side. For example, the rule {Light1 and Tv}=> {Light2} would suggest that light2 will be turned on whenever light1 and the television are on. Because there could be an enormous number of these rules in large data sets, there needs to be some measure of usefulness. This measure comes in the form of coverage and accuracy. Coverage describes the proportion of the data set that matches the features in the rule and accuracy expresses the chance that a rule is correct.

Training

Because there are potentially many rules, it is important to have a way of extracting useful rules efficiently. This algorithm is based on item sets. An item set is a collection of features that make up a rule. Item sets are different from association rules in the sense that there is no distinction between the left and right side in an item set. These sets combine features that can later be turned into a rule. The item itself is simply a feature-value combination that occurred in the data set.

There are two stages in the rule finding algorithm: generating item sets with a minimum amount of coverage, and from each item set determining the rules that have a minimum accuracy.

The first stage starts with finding the one-item sets with enough coverage. A one-item set is composed out of a single feature-value combination. These one-item sets are meant to eliminate all items that have a low coverage to begin with. This is necessary because larger item sets will be combinations of these one-item sets and these can only have a high enough coverage if each of the one-item sets has enough coverage. With these one-item sets, two-item sets are generated and the process is repeated to generate three-item sets and so on.

The second stage transforms the item sets into rules, checking whether they have enough accuracy. Generating rules from an item set is a matter of considering each item as the consequent of the rule. A three-item set {A,B,C}, for example, would produce three rules: $\{A \cap B \rightarrow C\}$,$\{A \cap C \rightarrow B\}$ and $\{C \cap B \rightarrow A\}$. The accuracy is calculated for each of these rules by dividing the coverage of the entire set by the coverage of the consequent. Rules that do not achieve the minimal accuracy are then discarded.

Mapping
The training set is a collection of recorded states. These states hold the values for each of the features of a room. From this training set, association rules are generated that relate features to each other. There could, for example, be the following recorded states:

Feature	State 1	State 2	State 3
Light1	on	on	on
Light2	on	on	on
Television	off	on	on
Time	19:55:00	20:00:00	20:05:00

Even though this is a small training set, it is possible to extract at least three useful association rules:

- If the television is on and light2 is on, light1 will also be on.
- If the television is on and light1 is on, light2 will also be on.
- If both lights are one, there is a good chance that the television will be on.

These rules can be used to make a prediction about future behavior. When both lights are on, it can be expected that the television will go on next, with an accuracy of 66%. What this example lacks, it a way to incorporate time into its predictions. Each state has a different time value so the coverage for each of these values is very low. The time feature will therefore not be considered in the rule extracting process.

This is where fuzzy logic may provide a solution. If time is divided into several fuzzy sets, corresponding to features, the value would be the membership of time to this set. This makes it possible to relate time sets to other features. In the example above, it would be useful to relate the evening to the lights being turned on. Adding fuzzy time sets to the same example may result in the following states:

Feature	State 1	State 2	State 3
Light1	on	on	on
Light2	on	on	on
Television	off	on	on
Morning	0	0	0
Afternoon	0	0	0
Evening	0.95	0.96	0.97

Chapter3 Transition probability

A suitable machine learning technique is required to capture and predict user habits. Such a technique requires a model that accurately defines user behavior. Many existing techniques and models have been evaluated in earlier research as described in the state of the art section. However, none of them seem to be optimal for our particular problem. This is why we designed a new technique that does not share the drawbacks of the other techniques. It is called transition probability and even though it is based on some of these existing techniques, it has its own unique approach to model and predict user habits. This chapter describes the idea behind transition probability and gives a detailed description of its operation.

Section3.1 Inspiration

Each machine learning technique has its advantages and disadvantages. In some cases, one approach excels in precisely the areas where others fail, and visa verse. This is the case for two techniques in particular; nearest neighbor classification and Markov chains. Both of these are suboptimal on their own but combining the two may improve their weaknesses. Transition probability is an attempt towards such a construction. The following sections give an overview of both techniques and provide an insight into how and why these different approaches are combined to form a new technique.

Section3.1.1 Nearest neighbor classification

Classification with a nearest neighbor algorithm is based on comparing a new sample with every other sample in a training set. This comparison results in a distance that represents the amount of equality between the new sample and the samples collected earlier. The new sample will be classified in the same class as the sample with the smallest distance.

Applied to the problem of predicting user habits, the samples will be snapshots of the input at a certain time. These samples are chronologically stored in memory. New samples are compared to these stored samples to determine the distance or inequality between them. Distance is based on the differences in time and input state.

Although this is a useful strategy in many classification problems, it has a major drawback in this context. The samples are taken at a certain interval and capture behavior within this time frame. What is not recorded, however, is the order in which samples have occurred. A prediction will indicate which samples are nearest, but a sample ahead in time may be just as near as a sample back in time. This makes it difficult to distinguish between something that will happen in the near future and something that has happened in the past.

The big strength of nearest neighbor classification is that it is able to learn from similar situations. If a new set of samples is different, but very similar to a set of samples recorder earlier, these sets will be considered to be from the same class. Although the small differences will cause some uncertainty, the new samples are appropriately classified as being the same as the older set of samples. This means that this technique does not require that a situation occurs before, to be able to predict it in the future. In a practical every day setting this is considered a big strength because new and unknown situations occur on a regular basis.

As described earlier, transition probability aims to combine the advantages of the techniques described above and minimize the disadvantages. For nearest neighbor classification this means that the ability to cope with unknown situations should remain and the unordered samples should be cast away. This requires a second technique that captures the order in which events took place but may be less effective with unknown situations.

Section3.1.2 Markov chains

Markov chains describe states and their transitions to other states. Each transition occurs with some probability that is solely based on the present state and is unaffected by any past states. With this chain it is possible to model a process whose state changes randomly. Because of this randomness it is impossible to predict the exact state in the future. The value of Markov chains lies in the statistical properties that can be extracted from the model and in some applications this is all that is required.

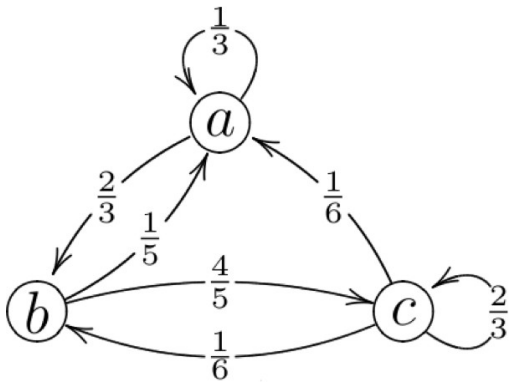


Figure 3.1.2.1: An example of a Markov chain

In order to describe a process in a Markov chain, there needs to be a definition of a state. In this case, this state is defined as one possible configuration of all inputs. These inputs may be the states of appliances, day of the week, time of day and the state of any streaming data. Transitions occur periodically and reflect the changes in activity of the user. The states in addition to recorded transitions, make up a Markov chain.

What makes this a powerful model for behavior prediction is that it is possible to look multiple steps ahead. This is made possible by the way that every action, or state transition, is stored in a particular order. A path through the chain traverses the states in a chronological order. This is why the set of reachable states from the current state, represent all the possible futures. Of course some states are more likely to be reached than others but it is likely that the intersection of all these states is non-empty, meaning that there are common features that will occur regardless. These common features are quite useful in dealing with unordered events. Unordered events are not uncommon in every day living situations. When certain actions take place in ever changing orders, these events are considered unordered. Predicting the order of these actions in future situations would prove to be difficult. A prediction based on Markov chains, however, is not concerned with the exact order in which states are reached because it is able to look beyond the individual actions and focus solely on the end result.

The weakness of Markov chains lays in the fact that every prediction is based on the current state and the transitions from this state. Whatever the current state is, it needs to exactly match a state in the model. If the current state was unreached until this moment, no transitions from this state are known and no prediction can be made about future states. The inability to handle new situations based on previous similar situations makes this technique somewhat impractical in real life. New situations will occur regularly and will render the model inadequate.

These features are the exact opposite of the features found in nearest neighbor classification where unknown states are dealt with with ease and orderings pose an insurmountable problem. The following sections give insight into how the positive features of these two techniques can be combined whilst excluding the negative features.

Section3.2 Model of the environment

At the base of transition probability, resides a model that translates the real world in a usable data structure. The design goal for this model is to create a structure that is small and simple, yet provides the prediction algorithm with all the information it requires. Storing too much information, however, imposes an unnecessary burden on memory. Our model is constructed using three basic components; states, transitions and sectors. These components will be described in the following sections.

Section3.2.1 States

The constant monitoring of a household provides a set of values, each belonging to one of the input features. One unique combination of all of these values and features is represented in a single state. Our definition of a state, therefore, includes one main attribute that stores the values for each of the features. This is a smaller and simpler perspective on what a state should be compared to the one needed for the application of Markov chains as described earlier. In our model, time is no longer included as a feature, which dramatically scales down the total state space and overall complexity. Representation of time in our approach will be discussed in section 3.2.2.

A second important aspect of a state is the definition of similarity between states. This concept is borrowed from nearest neighbor classification and is crucial for the ability to learn from similar situations when a new and unknown situation arises. The prediction will, in this case, be based on previous occurrences of similar events. However, some situations will resemble the present more than others and should, therefore, have more influence in the prediction process.
Similarity is defined as a summation over the difference between each feature of two states. $V_A(f)$ represents the value of feature *f* in state *A,* and R(f) equals the total range of possible values of *f.*
The similarity of *f* in states *A* and *B* is calculated as $1 - (V_A(f) - V_B(f)) / R(f)$. A summation over all features gives a measure of how similar states *A* and *B* are.

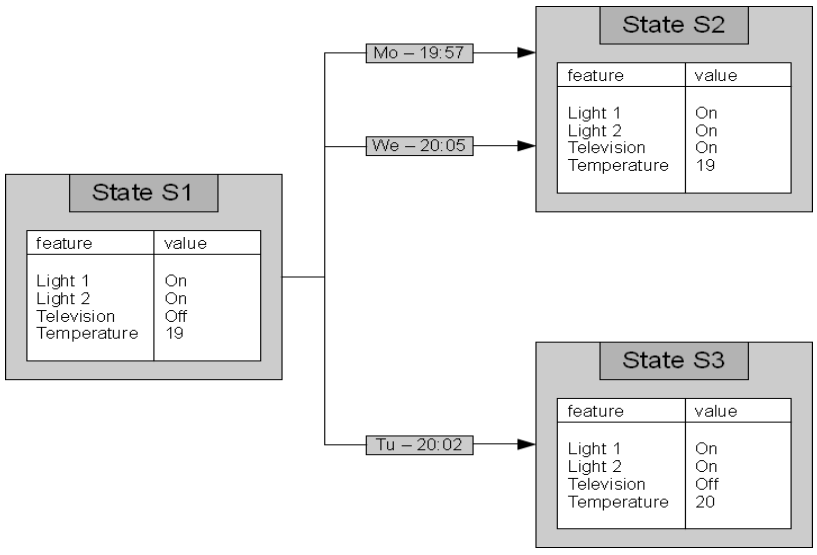


Figure 3.2.1.1: An example of multiple states and the transitions between them

Section3.2.2 Transitions

Whenever something changes in the users environment, the current state no longer represents the current situation. A state transition captures this event and connects two states to one another as depicted in figure 3.2. The exact time and date are stored in the transition data structure to be recovered later by the prediction algorithm. This has a significant advantage over a model where time is stored within a state. Time no longer needs to be divided into fixed intervals to keep the model consistent. Any limitations on the accuracy of time representation are thereby eliminated , this will result in a simple, yet accurate model.

Section3.2.3 Sectors

The number of different states and transitions may grow rapidly when the number of input features increases. More states are needed to cover every feature-value combination and with more inputs, it is more likely that something changes and a transition occurs. This increase in data is a burden on memory and, more importantly, requires more computation to perform a prediction. More states and transitions need to be considered to get a clear view on possible future events. However, not every transition is equally important when it comes to predicting a certain action. For example, if a light is turned on in the kitchen, it is not unlikely that this will have no effect on whatever is going on in the living room. In this case, it would be unnecessary to include the kitchen light as an input feature for predictions concerning the living room.

We avoid unnecessary interference by adding sectors to the model. A sector captures all events that took place within the boundaries of the physical space it represents. Every sector has its own collection of states and transitions, influenced solely by the input features residing in it. A consequence of this is that predictions are made per sector and only useful events are considered. This makes the algorithm run faster and causes the entire system to be more scalable.

A strict separation of input features per sector may lead to inaccurate predictions when an action spans multiple sectors. Events occurring in the living room might have been triggered by something that happened in the kitchen. It would not be possible to predict this since predictions concerning the living room do not incorporate input features from the kitchen. This is why sectors do not always need to be disjoint from each other and overlaps are supported, which may provide a smoother transition.

Section3.3 Prediction algorithm

Our prediction algorithm is divided into three steps; initialization, influence distribution, and a final processing phase. The following sections are dedicated to each of these steps and a final section formulates their interconnections in the form of a pseudo code.

Section3.3.1 Initialization

After acquiring a sufficient data set, it is possible to start making predictions about future events. At this point, memory is filled with a substantial amount of states and transitions providing a detailed history of user activity. It is not necessary, however, to involve the complete data set into every prediction. The separation between important and non-important data is based on two parameters; prediction time *t* and prediction range *r*. When a forecast is requested for time *t* and needs to be valid for range *r*, a subset is extracted from the data set containing all the events that occurred between *t* - *r* and *t* + *r*. This includes all the transitions within the range, and all the states that were reached in this time frame. A prediction based on this subset gives a forecast on the events that will occur between *t* and the end of the specified time slice. Note that prediction time *t* is specified as a time of day and lays within the range of 24 hours. This means that the data subset can cover multiple days of events occurring within the given time range.

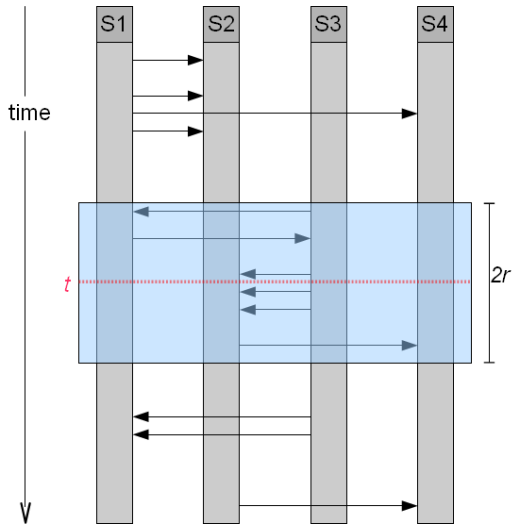


Figure 3.3.1.1: An example of preselection of transitions within range

The process that follows the preselection of states and transitions is based on an influence distribution system. Each state receives an initial amount of influence, quantified in points, based on their similarity to the current state and the number of occurrences before the start of the range. Similarity between states has been described earlier and is a simple function that rates the similarity of two states with a number between 0 and 1. This will cause highly similar states, or situations, to receive more initial points. The second parameter in the initial point distribution is acquired by counting the number of times a state has previously occurred. For each day in the data set, it is determined what the last state just before the start of the range was. A summation over all days, results in a total occurrence count per state. By multiplying the similarity with the number of these occurrences, more points will be rewarded to states that are both common and similar to the current state.

Section3.3.2 Influence distribution

After assigning initial points, or influence, to each of the states in range, points are redistributed according to transitions between states. Only the transitions selected by the previous step will be used in this process. An important metric in the distribution process is the number of times a state is reached. This is different from the number of occurrences, which only focuses on what happened before the range. To achieve a correct distribution, it is also important to consider what happened within the range. If a state *S* has occurred *o* times and there are *i* transitions ending in *S*, this state is considered to be reached *o* + *i* times. The following transition set was extracted from the ones in figure 3.3.1.1:

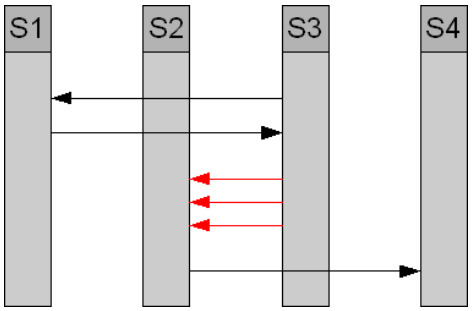


Figure 3.3.2.1: Simple point reassignment

This subset contains three transitions from S3 to S2. Therefore, if state S3 was reached *n* times, it will pass $3/n^{th}$ of its points to S2. By doing this, points are kept consistent with the chance of a state being reached in the future. If more transitions travel towards a particular state, this state is more likely to be reached and should therefore receive a larger amount of points. These points are taken away from the origin of the transitions, capturing the decrease in probability of future occurrence.

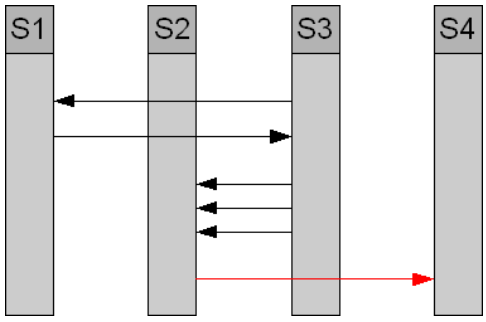


Figure 3.3.2.2: Recursive point reassignment

Unfortunately for S2, this is a recursive algorithm. If this state is reached m times, $1/m^{th}$ of any points granted to S2 will be redistributed through the transition marked in red, including the initial points. This means that S4 indirectly benefits from the relation between S3 and S2.

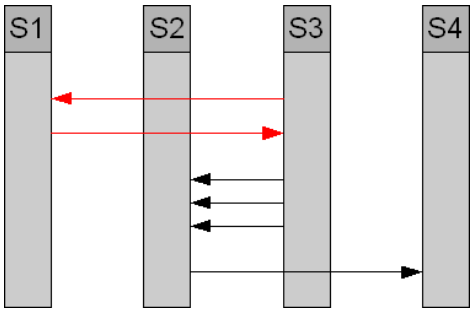


Figure 3.3.2.3: A loop

Figure 3.3.2.3 illustrates a special configuration of transitions; a loop. Strictly following the recursive distribution algorithm would result in an infinite loop of passing around points between S1 and S3. This is why points become immobile once they have returned to state visited earlier. Only after rejoining their original state will these points cease to redistribute. Stopping this process one step earlier, and thereby ending up in the last unvisited state, would cause the points of S1 and S3 to be swapped. Such a single swap will represent the exact opposite of the actual probabilities.

Section3.3.3 Final processing

Once all points have reached their final destination, the results can be processed to form a prediction of future behavior. A prediction assigns probabilities to features having a certain value. Each of these feature-value combinations is related to a number between one and zero, representing the probability that this feature will assume the value of this combination. What is known from point distribution is the likelihood of every state. Feature-value combinations have received an equal amount of points as the states they are contained in. Because combinations can be represented in multiple states, their total amount of points is a summation of the points of these states. Dividing these totals by all the points in the current state space, gives probability of this feature-value combination.

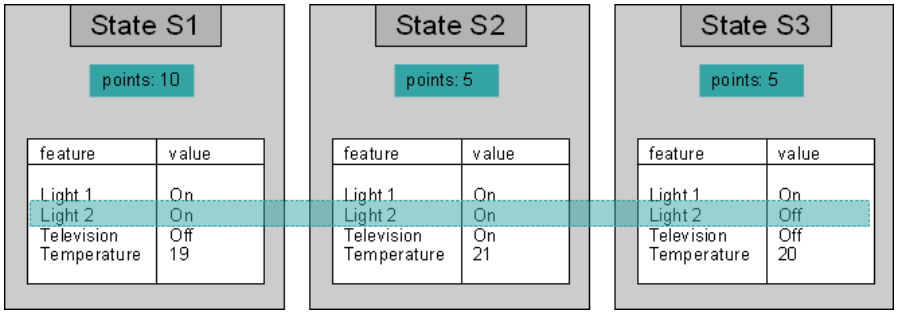


Figure 3.3.3.1: A point processing example

Assuming that figure 3.3.3.1 represents the entire state space, light2 would have a fairly high chance of being on. Calculating exactly how high this chance is, is a matter of adding the points of all the states wherein light2 is on, and dividing this number by the total number of points in the state space. In this case, 10 points from S1 plus 5 points from S2, divided by 20. This results in a probability of 75 percent, and automatically, a 25 percent chance that light2 will be off since this is a binary input device. Repeating this process for every other input results in the following table;

Feature	Value	Probability
Light 1	On	1
	Off	0
Light2	On	0.75
	Off	0.25
Television	On	0.25
	Off	0.75
Temp	19	0.5
	20	0.25
	21	0.25

Table 3.3

Every feature-value combination is assigned a probability, completing the prediction process. The system is now able to make decisions based on these numbers and provide energy saving services to the user.

Section3.3.4 Pseudocode

```
Prediction getPrediction(prediction time Pt, prediction range Pr){
    //preselection
    for every (Transition t in all recorded transitions){
        if(t.time is between  Pt - Pr and  Pt +  Pr){
            select t;
            select t.startState;
            select t.endState;
        }
    }
    //calculate number of occurrences per state
    for(Day d = first day ; d before current day ; d = next day){
        State s = the state at time Pt – Pr on day d;
        occurrences(s)++ ;
    }
    //calculate number of times reached per state
    //init
    for every(selected state s){
        reached(s) = occurrences(s);
    }
    //count
    for every(selected transition t){
        reached(t.endState)++;
    }

    //distribute points
    //init
    for every(selected state s){
        result(s) = 0;
    }
    //distribute
    total amount of points Tp = 0;
    for every(selected state s){
        initial points = currentState.similarityTo(s) * occurences(s);
        CALL distribute(s ,initial points,result);
        Tp += initial points;
    }
```

```
        //final processing of result
        //init
        for every(feature-value combination fv){
                prediction(fv)=0;
        }
        //process
        for every(selected state s){
                for every(feature-value combination fv contained in s){
                        prediction(fv) += result(s) / Tp;
                }
        }
        return prediction;
}

void distribute(current state, points,result){
        keep = points;
        for every(Transition t in selected transitions){
                if( current state == t.startState AND NOT t already processed ){
                        give = points / reached(current state) * time distribution(t.time);
                        keep -= give
                        CALL distribute(t.endState,give,result)
                }
        }
        result(current state) += keep
}
```

Section3.4 Relevance distribution over time

As described in the algorithm section, relevant transitions are selected based on their distance from the prediction time. The reason for this is that transitions in this slice of time are more likely to repeat themselves than other transitions, and are therefore considered to be more relevant for a prediction. This relevance function, however, is very coarse grained. A transition is either relevant or not and transition at the edge of range are just as important as the ones at the center. More gradience is needed to differentiate between relevant transitions and to better reflect the actual chance of a transition reoccurring at a certain time. This calls for a function to distribute relevance according to differences in time. The relevance of a transition is always based on its distance to prediction time *t*. Any function that maps relevance to this distance could be used for the algorithm.

Originally, if there was a transition T from state S1 to S2, and S1 was reached *n* times, then $1/n^{th}$ of the points received by S1 would be passed to S2. Including time distribution in the algorithm requires one additional multiplication. Transition T has a relevance *r* between one and zero. If *r* is high, more points should be passed to S2 than when *r* is low. The higher the relevance, the more chance that this transition will occur and the more chance that S2 will be reached. This idea is captured by multiplying $1/n$ with *r* and redistributing a proportional amount of points.

Chapter4 Implementation

So far the focus has been on the theoretical aspects of transition probability. If this technique is to be employed in a practical setting, it is important to test it in a simulated environment first. The following sections will give an insight on the overall structure and functionality of the implementation.

Section4.1 Architecture

The system is composed of several modules, all with their own specific function and purpose. This section is intended to give an overview of these components and functions without going into much technical details. An overview of the modules and their relation to each other is presented below.

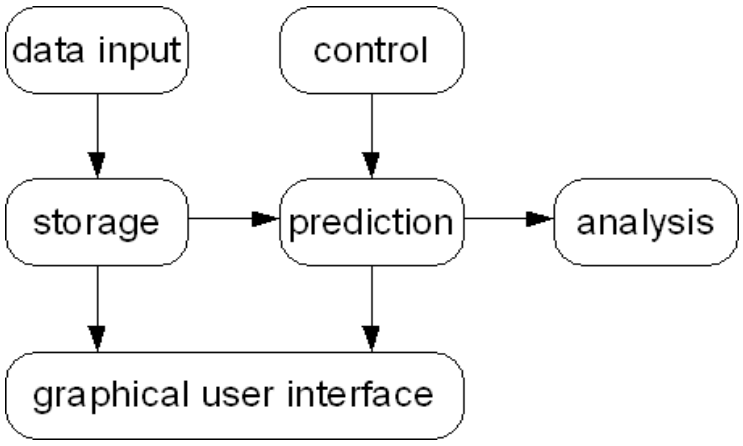


Figure 4.1.1: System architecture

- **Data input**
  All predictions are based on recorded behavior of the past. The data set containing this information could come from anywhere and is presented in its own format. This includes sensor identification and a way of representing sensor values. Data can be stored in a single file or in a more structured, hierarchical set of folders and files. Whatever the origin and format may be, all data needs to be converted to fit inside the model of states, rooms, and transitions. The input module facilitates this conversion and makes it possible to fill the system with a useable data set.

- **Storage**
  The storage module acts as a storage space for input data. This data is stored per room and has taken the form of states and transitions. From here, other modules can access the data in a uniform format.

- **Prediction**
  Here, the acquired data is processed into a prediction using an implementation of the transitions probability algorithm. How exactly the algorithm is implemented is described in section 4.3.

- **Analysis**
  For the purpose of evaluating the performance of the algorithm, the analysis module contains the functionality required to get a measure of performance and accuracy. A more detailed description of this functionality is given in chapter 5

- **Control**
  Rules regarding how and when predictions should be made, are implemented within this module. These rules involve some parameters further explained in section 4.2.

- **Graphical user interface**
  Both the predicted user behavior as the actual behavior are presented side by side in a graphical interface in order to easily compare the two. A separate interface, specifically designed for debugging, shows the internal process for each prediction. The user interface is discussed in more detail in section 4.3.

Section4.2 Functionality

One purpose of this implementation is to find out how the algorithm can be applied in a practical context. The ultimate practical context would be an actual house with real inhabitants living their lives with the aid of Go-Green technology. This requires an up to date perspective on future events and the ability to react to an ever changing environment. Rules that define when and how predictions should be made are needed. These rules are constructed using three parameters; interval, range, and threshold.

Meeting the expectations about the future up to date requires regular predictions. Exactly how regular these predictions need to be, is specified with the interval parameter. The interval defines the maximum time difference between two consecutive predictions. Ideally, the interval is as short as possible so that the system is as up to date as possible. More predictions, however, cost more memory and more computations so there is a trade off here. Interval is defined as the maximum time between predictions. This maximum is not always reached because there is another reason, besides the passing of time, why an updated prediction is required. A state transition also requires a new prediction. When the current state has changed, the last prediction becomes obsolete because it is based on an old view of the present. A new prediction is made when either an interval of time has passed, or the current state has changed.

As described in chapter 3, the prediction range plays an important role in the prediction process. Only transitions that occurred within this period have an affect on the outcome. This parameter can be used to investigate the influence of different range settings on the performance. Not only does it affect the accuracy in which the future is predicted, it also has an impact on memory and processing requirements.

The threshold parameter is necessary due to the probabilistic nature of the algorithm. Whether to turn a device on is a binary decision. A prediction only gives the probability that this device needs to be turned on so this probability needs to be converted to a binary value. This can be achieved by setting a threshold. If a forecast about a certain feature-value combination has a higher probability than this threshold, it will be considered “true”. Changing the value of this parameter has some effects on the way decisions will be made. Setting the threshold higher makes it more likely that a prediction is considered false, thereby causing it to be discarded. This, in turn, causes the system to be more conservative in its decision making. Implementing this parameter makes it possible to experiment with different threshold settings and to evaluate its influence.

Section4.3 Graphical user interface

Our implementation of transition probability features a graphical user interface designed to present the inputs and outputs of our algorithm and to provide useful debugging information. The output window, depicted below, allows the user to compare sensor values with predicted values for a certain day. Figure 4.3.1 shows the sensor values and predictions for "go to bed" on February the 19th.

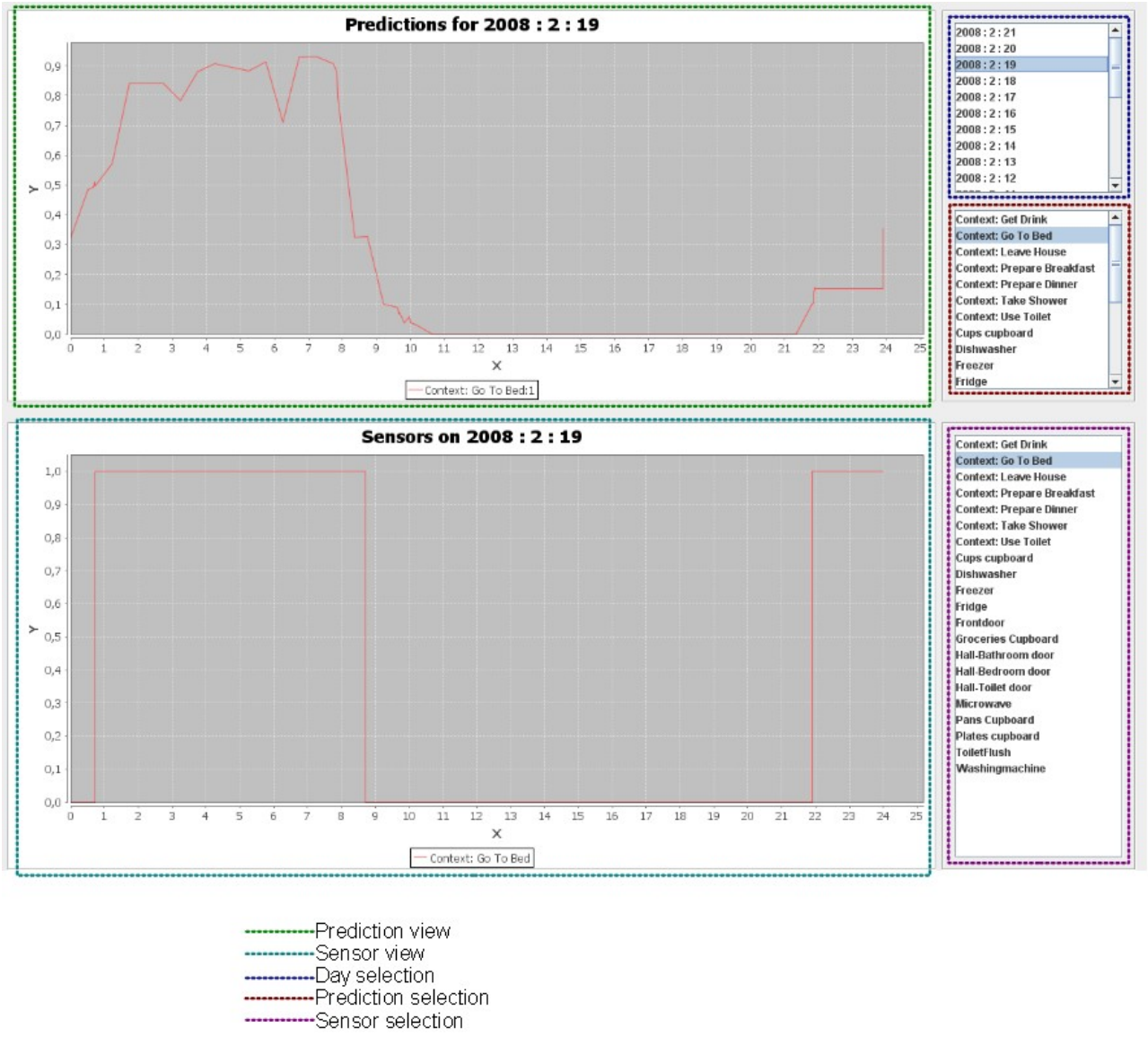


Figure 4.3.1: Output window

Figure 4.3.2 shows the debugging window for a prediction made on the 25th of January at 0:20. The state space on the left shows every state and transitions involved with this particular prediction. Selected state s4 is marked in blue and its properties are displayed in the selection info panel. Green and red transitions are transitions towards and from s4 respectively. All transitions are vertically ordered, from early to late, based on their time of occurrence. State s0, which is the current state for this prediction, is displayed in orange.

The selection info panel, currently set to state s4, shows the number of points assigned to s4, its similarity to the current state, the number of times s4 has been reached, the number of occurrences, and the feature-value combinations that make up this state.
Prediction results, displayed on the right, show the probabilities for every value of every features.

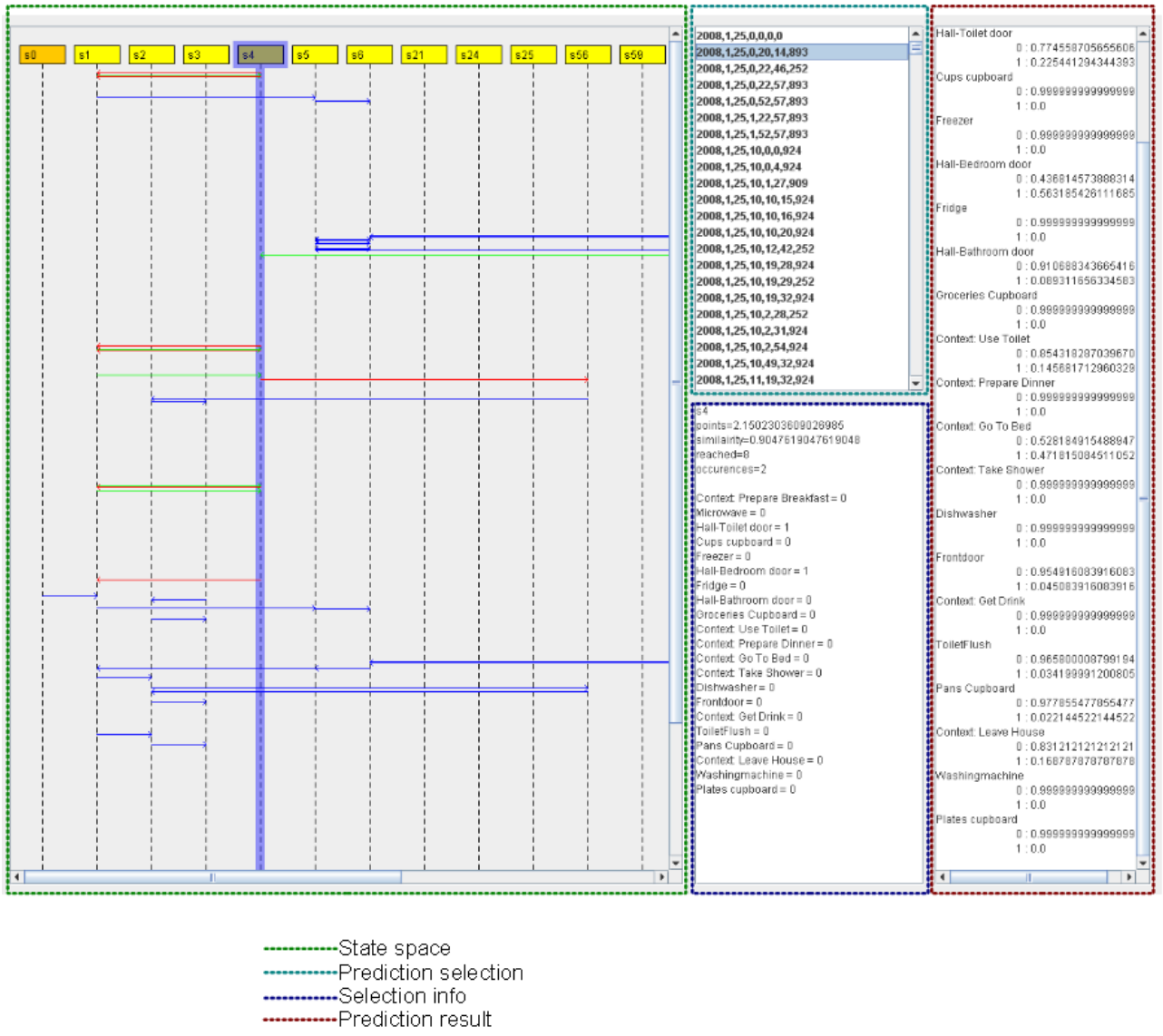


Figure 4.3.2: Debugging window

Section4.4 Algorithm implementation and optimization

The algorithm is defined in a way that does not leave any room for interpretations. All steps are fixed and an implementation is required to include these steps exactly as they are described. There is, however, some degree of freedom in the relevance distribution used to acquire a fine grained measure of importance of transitions. This distribution function is only expected to assign an importance to a transition based on its time of occurrence. Many functions could fulfill this role and finding the optimal one could be difficult. For this implementation, a linear function is chosen to act as a distribution function.

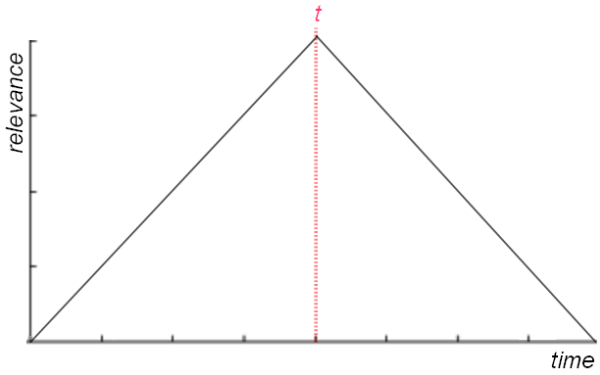


Figure 4.4.1: Linear relevance distribution

As illustrated in figure 4.3.1, relevance peaks at prediction time *t* and degrades linearly over time. The simplicity of this function makes it highly suitable for an initial implementation where there is little known about how the system should behave. Additionally, no methods are available to distinguish an optimal time distribution from a sub-optimal one. The fact that this is a relatively simple function does not mean that it cannot produce an optimal distribution.

Even though the steps in the algorithm are fixed, there is some room for optimization in the implementation of one of these steps. The point distribution system redistributes the points granted to states, based on the transitions between them. When there are *n* transitions between states S1 and S2, the distribution function needs to be called at least *n* times to move the required amount of points to S2. This is because the distribution handles one path at a time. If *m* transitions go from S2 to another state, then that would result in $n \cdot m$ different paths which are all handled separately.

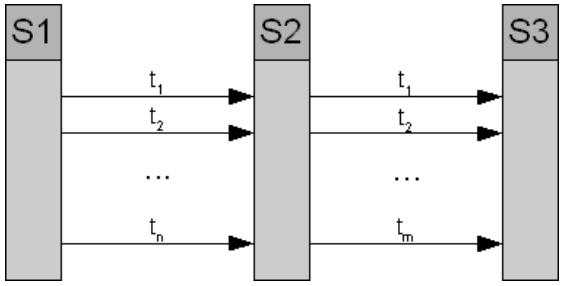


Figure 4.4.2 : Distribution based on transitions

We perform an optimization to minimize the number of paths used in the point distribution step. All transitions between two states are combined to form a single bundle of transitions. This bundle represents the combined probability that one of the bundled transitions will occur. The combined probability is no more than a summation over all the individual probabilities of transitioning. Point distribution now evaluates bundles instead of individual transitions. Applying this technique to the example above would result in the following situation.

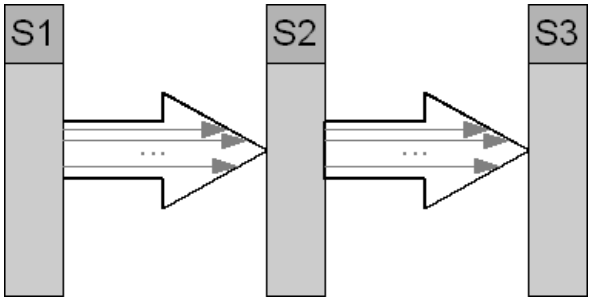


Figure 4.4.3: Distribution based on bundles

Performing this optimization results in significant reduction of the number of paths, thereby reducing the number of distribution steps. Since the point distribution step is the most computation intensive step of the algorithm, this optimization may have a positive effect on latency and processing requirements. Because the outcome of the prediction is unaffected, this optimization causes no negative side effects for other areas.

Chapter5 Evaluation

The primary goal of the implementation of transition probability as a learning technique is to facilitate an evaluation of the performance. Performance can mean many different things is many different situations. This chapter clarifies the definition of performance and describes the way different conditions affect performance. Additionally, a comparison is made between transition probability and several other learning techniques.

Section5.1 Data set

An evaluation of our learning technique requires a training set of labeled activities and sensors. User habits hidden in this data are to be uncovered by the designed algorithm. Because the activities are labeled, it is possible to verify any habits that are found. The success with which this is done is a good indication of performance. Multiple third party data sets [4] [15] [15][16] [17] were considered and evaluated on their applicability and usefulness .

The process of creating the data set we used is described in [17]. This data set is specifically designed to be used for activity recognition. Wireless sensor nodes were placed in a 3 room apartment to monitor a dozen appliances and utilities. The activities of a single inhabitant were recorded for one month. Every activity was labeled in real time using a blue-tooth headset and speech recognition. This resulted in an accurate set of labeled activities and sensor data, listed below:

Activities:

- Leave house
- Use toilet
- Take shower
- Go to bed
- Prepare breakfast
- Prepare dinner
- Get drink

Sensors:

- Microwave
- Hall-Toilet door
- Hall-Bathroom door
- Cups cupboard
- Fridge
- Plates cupboard
- Front door
- Toilet flush
- Freezer
- Pans cupboard
- Washing machine
- Groceries cupboard
- Hall-Bedroom door

What makes this data set useful for a first experiment with activity recognition is that it is a relatively small set. This also holds for the number of different activities that were recorded. Combining sensor data and activity labels, gives 22 inputs to the learning algorithm, making it quite manageable and uncomplicated.

The algorithm supports the separation of inputs into different rooms. This allows the prediction to be based on just one room, without any interference from events in other parts of the house. A division also links sensors to each other in a natural way. Linked sensors have a higher chance of affecting each other since they are in the same room.

Unfortunately, there is no description available about which sensors and activities are located in which room. Common sense could provide a reasonable layout of sensors and rooms, but there is another issue; Activities in this data set are not bound to just one room. The “leaving the house” activity, for example, may start in the kitchen and “move” into the hallway where the house is actually left. This is because the intention to leave the house is recorded instead of the actual action. A moving activity can not be bound to a single room, and it is impossible to bound an activity to multiple rooms since this may lead to contradictions. Therefore, the entire apartment is interpreted as a single room with all sensors and activities residing in it.

Section5.2 Performance metrics

Section5.2.1 Prediction accuracy

Not all predictions will be correct. Wrong or inaccurate predictions can jeopardize the objective to increase user comfort and to save energy. This makes prediction accuracy an important factor in the evaluation of a learning technique. Accuracy is measured by comparing the predicted events with the actual events in the data set. Two parameters, i.e, detection rate, and false alarm are used to express the difference between these two sets of events.

The detection rate reflects the probability that an actual event is predicted prior to its occurrence. This is implemented by parsing through all transitions, and checking for each of these if the last prediction before the transition time matches this event. Such an agreement implies that the transition was detected. Dividing the detected events by the total number of events, gives the detection rate.

Even though the detection rate is a useful parameter, it is not enough to completely define accuracy. An event, or state transition, is due to a change in the value of one of the inputs. Every time the input changes and the predictions agree with this change, the detection rate increases. If an algorithm always predicts change, regardless of the situation, the detection rate will be very high since all changes were “predicted”. This prediction strategy, however, can not be considered accurate because all predictions are wrong whenever the user is actually inactive. A good definition of accuracy therefore needs a second parameter, i.e, the false alarm that captures these false predictions in times of inactivity.

The false alarm works in the same way as the detection rate, only this time from the predictions point of view. Similar to input transitions, prediction transitions occur when two consecutive predictions give different views on the future. Prediction transitions are true when the actual behavior reflects what was predicted, or false otherwise. Dividing the number of false accounts by the total number of prediction transitions, gives the false alarm.

An optimal prediction accuracy has a high detection rate and a low false alarm. Detecting upcoming events makes it possible to better predict user habits. The more events are detected correctly, the more user comfort can be ensured by automation. A low false rate implies a high probability that predicted events will actually occur.

Section5.2.2 Latency

Real time behavior requires the system to react quickly to changing environments. However, a prediction requires a certain amount of time to make. The time it takes to make one prediction is defined as latency. In order to observe the changes in latency over a period of several months, individual prediction latencies are measured and averaged over every month.

Section5.2.3 Memory usage

It is important to know what kind of hardware is needed to run this algorithm, if a practical implementation is to be realized. Latency and memory usage are the main concerns and the focus will lay on these two measures. We will measure memory usage accurately using the Netbeans profiler tool.

Section5.3 Impact analysis of changing conditions

Performance is not a static characteristic and it depends on the context in which this system is operating. Several different conditions can be used to distinguish an optimal context from a suboptimal one. This chapter proposes a number of key conditions that are of importance to the performance of the prediction algorithm. Each condition is analyzed based on its impact to the performance measures described earlier.
For the purpose of a consistent analysis, algorithm parameters are set to default settings. Prediction range is set to 10 minutes, interval is set to 30 minutes and the threshold is set to 0.3. The impact of using different parameter settings will be analyzed separately.

Section5.3.1 Amount of training data

Prediction algorithms typically need to observe behavior several times before being able to accurately identify user habits. An increase in the amount of training data is therefore expected to cause an equal increase in accuracy. However, additional training data requires more memory to store it in and more processing to evaluate it.

The effects of the amount of available training data are made clear by copying existing data and evaluating after each step. Copying data produces a larger artificial training set that contains multiple instances of the same events. Such a set is useful when purely the effects of more data are to be measured without interference from changes in user behavior. This is not, however, comparable to an unaltered recording of user behavior over a longer period of time.

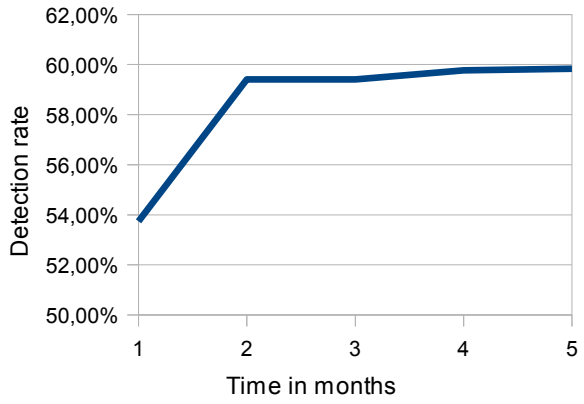


Figure 5.3.1.1: Detection rate

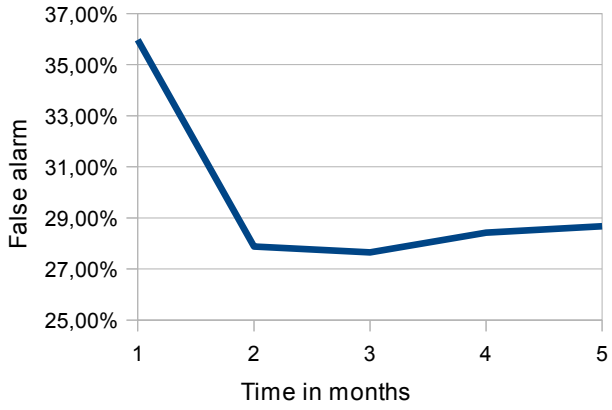


Figure 5.3.1.2: False alarm

Accuracy
Diagrams above illustrate the effects of a growing data set on the detection rate and false alarms. All accuracy measurements are based on the average rates over the activity features and span a total of 5 months. Both the detection rate and the false alarm improve rapidly in the first 2 months. Habits become more apparent and the transition flows through the states gain in strength. The symmetry between these improvements shows that learning has an equally positive effect on both accuracy parameters. After 2 months, the system has learned all it can learn. Mistakes are repeated and, in some cases, become so persistent that the false alarm starts to increase again.

Latency

As more data becomes available, predictions become more time consuming. More transitions exponentially increase the number of paths between states. Figure 5.3.1.3 displays the latency increase over a period of 5 months.

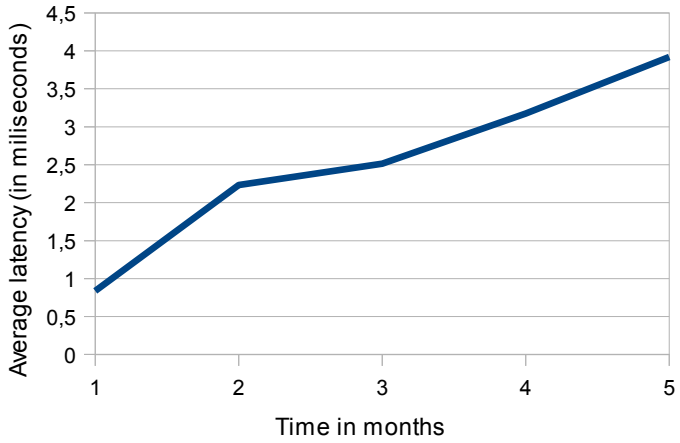


Figure 5.3.1.3: Average latency

Latency increase behaves linearly even though distribution paths increase exponentially. This is due to the use of bundles, instead of individual transitions, in the point distribution process. The number of bundles remains unchanged when multiple copies of the same data are used.

Memory

The increase in memory usage over time is also linear. Causing this, is the equally linear increase of transitions due to data copying.

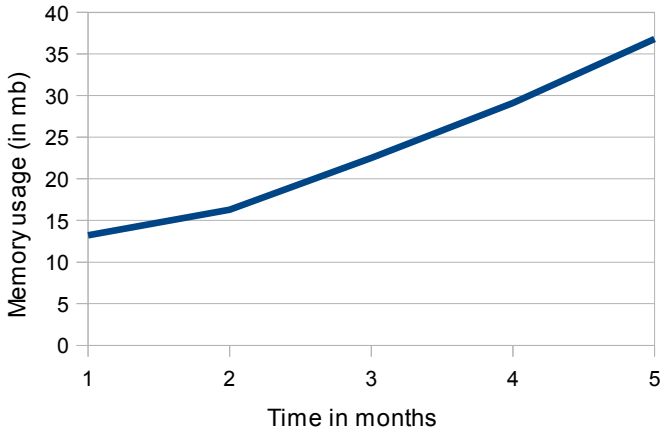


Figure 5.3.1.4: Memory usage

Section5.3.2 Level of variance in behavior

An impact analysis on the amount of the training data revealed that accuracy increases when more data is available. This analysis was based on duplicating one month of data to create several months of the same data. Duplication brings about a high level of predictability and ideal learning circumstances. Real world circumstances are likely to be far less ideal. User behaviour typically exhibits a high level of variance and includes unpredictable actions that do not conform to any habit. In order to analyse the impact of variance in user behaviour with the current data set, random time shifts are introduced in every copy of the original data. Time shifts are of a random size, within a certain limit.

Accuracy

A higher level of variance has certain affects on the activity detection rate. Depicted below is the chart illustrating the detection rates over a period of 5 months with time shifts of 0,15,30,45,60, and 120 minutes.

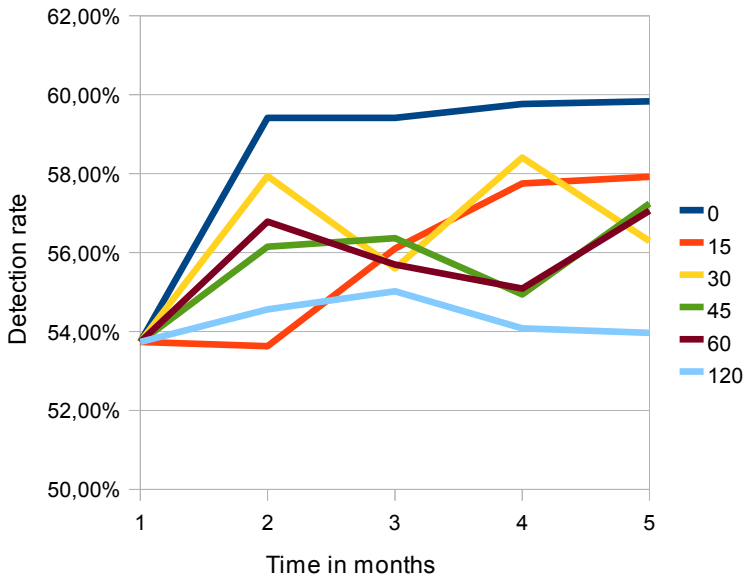


Figure 5.3.2.1: Detection rates

As the level of variance becomes higher, the detection rate steadily becomes lower. It gets increasingly difficult to find patterns in user behaviour when activities become scattered throughout the day. With time shifts of a maximum of 2 hours, correct detections are dominated by events that mark the end of an activity. When transitions are more spread out, it becomes more likely that an activity will not occur at all, or will stop immediately, because these states become more and more common in any given prediction range. This is the reason why detection rates are creeping towards 50 percent, and only stopping events are correctly detected.

Figure 5.3.2.2 displays the false alarms at which predictions are done over a period of 5 months.

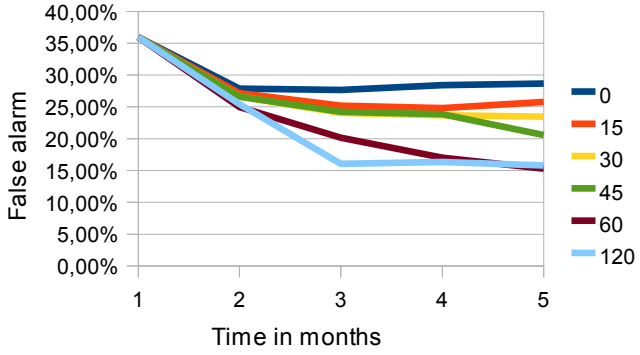


Figure 5.3.2.2: False alarms

A higher level of variance causes false alarms to decrease. This seems counter intuitive because this would actually increase overall accuracy. Whenever an activity is predicted, it becomes more and more likely that this predictions is correct as variance increases. The reason for this is also related to the scattering of transitions and the increased probability of an activity not occurring or stopping. When an activity starts, this has a low probability when the variance is high. This makes the algorithm behave very conservative when it comes to the prediction of an activity starting. Conservatism causes less false alarms but also decreases the detection rate.

Latency

The figure below is structured differently from the accuracy charts to illustrate an unexpected effect of variance on prediction latency. Time shift ranges are on the x-axis and each line represents the time span in months.

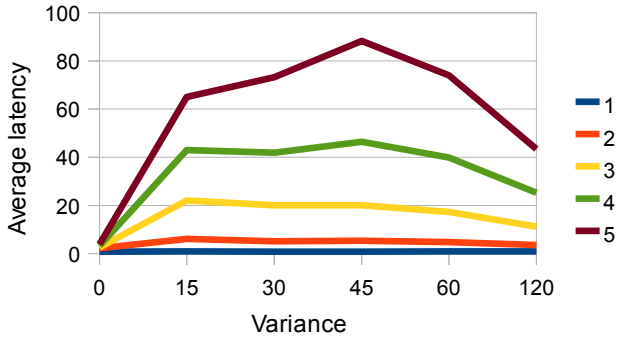


Figure 5.3.2.3: Latency

What is remarkable about this chart is that latency first increases and then decreases as variance gets higher. The initial increase is caused by different activities interfering with each other. Transitions in a single prediction range can come from multiple activities because of their time shifts. This causes the application of bundles, which are used as an optimization on the standard algorithm (section 4.3), to be less effective because only transitions between the same states can be merged into a bundle. Due to an increase in the number of bundles, latency gets exponentially higher.

Latency decreases when variance grows beyond 45 minutes. With these high variances, transitions get spread out so much that individual predictions cover smaller amounts of transitions, thereby decreasing the number of bundles.

Section5.3.3 Different types of activities

Previous impact analyses used average values over all activities to explain global impacts. By generalizing the impact, individual attributes of an activity are ignored. Some activities, for instance, make a greater contribution to overall accuracy than others. Accuracy is the most important performance factor for this impact analysis because latency and memory usage are unaffected by the presence of certain types of activities. These factors are only influences by the amount of activities.

Accuracy

Depicted below are the individual detection rates for every activity.

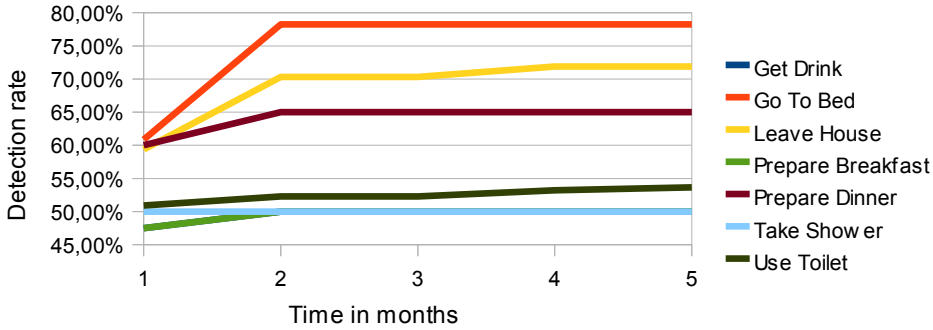


Figure 5.3.3.1: Detection rates

Activities with a long duration like “go to bed” and “leave house” tend to have a higher detection rate than short activities. This is partly due to the fact that these activities are more regular in their time of occurrence, but even more due to their duration. Transitions marking the start and end of an activity are closer together for short activities, thereby increasing the chance that both are within the prediction range. When these transitions cancel each other out, it becomes more difficult to see whether the activity is about the start or if it has already finished. In this situation, the prediction will be consistent with the state that is most often reached at the time of predicting. Because the short activities are also more irregular, the most common state does not include this activity. Most predictions forecast that the activity will not occur at all, or stop when it does. The detection rates of around 50 percent are due the the detections of an activity stopping.

False alarms behave in very much the same way. Predictions about short activities describe a future in which the activity does not occur. For most of the time, these predictions are correct, leading to low false alarms. Activities with a long duration are better detected but also have a higher false alarm because of time variations, and sudden non occurrences.

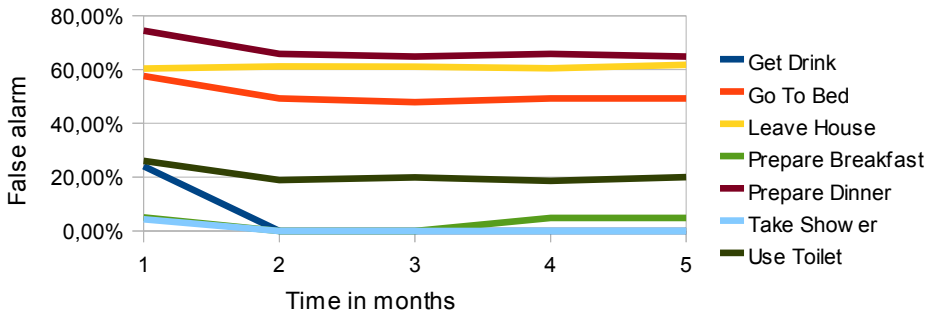


Figure 5.3.3.2: False alarms

Section5.3.4 Different algorithm parameters

So far, impact analyses used fixed internal settings to study the effects of external conditions on performance. Default parameters defined "standard" prediction behaviour. These settings, however, are by no means optimal and many other possibilities are left unexplored. The contribution to accuracy will be analysed for each parameter separately.

Prediction interval

Predictions are made whenever a transition occurs and the current state has been altered. Additionally, if the time since the last prediction has exceeded a predefined interval, the forecast for the future is updated. Setting the interval to different values, in this case 5,15,30 and 45 minutes, results in different levels of accuracy;

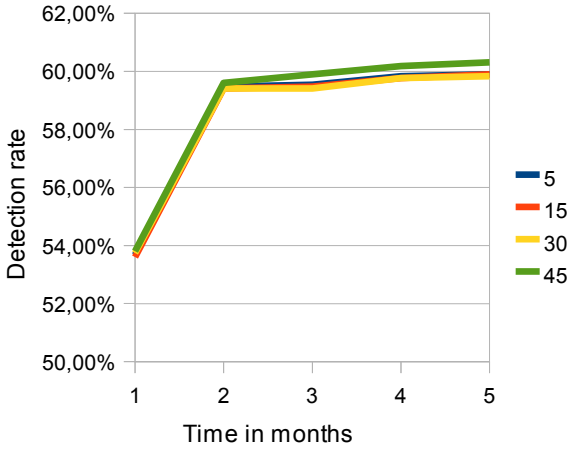


Figure 5.3.4.1: Detection rates

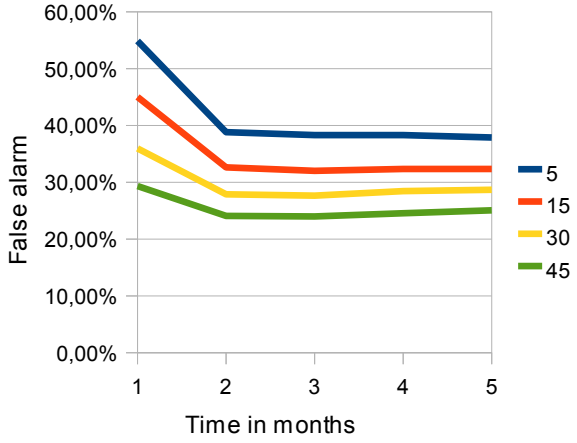


Figure 5.3.4.2: False alarms

Intuitively, smaller intervals are likely to produce higher detection rates because more predictions are made. This was an important reason for implementing such a parameter. Measurements show however, that the interval has little or no effect on the detection rate. The reason for this is that the recorded behaviour is very bursty. There are either many consecutive transitions close together, or long periods of no activity. Because every transition triggers a new prediction, busy periods also feature the highest density of predictions, causing high detection rates. It can be concluded from this that is it sufficient to only make new predictions when a transition occurs regardless of the time since the last one.

False alarms steadily decrease as the predictions interval increases. A short interval causes more predictions to be made at times of no activity. This increases the chance of a false alarm because these predictions are always wrong when they forecast an activity.

Prediction range

All transitions within the prediction range are considered in the predicting process. A larger range contains more transitions and contains a more complete view activities around the time of predicting. When the range becomes too large however, this view is distorted by events further away in time that are of no consequence to the current situation. The charts below describe the accuracy of prediction range between 5 and 30 minutes.

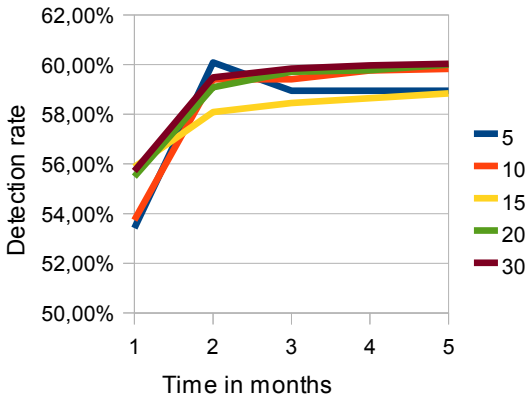


Figure 5.3.4.3: Detection rates

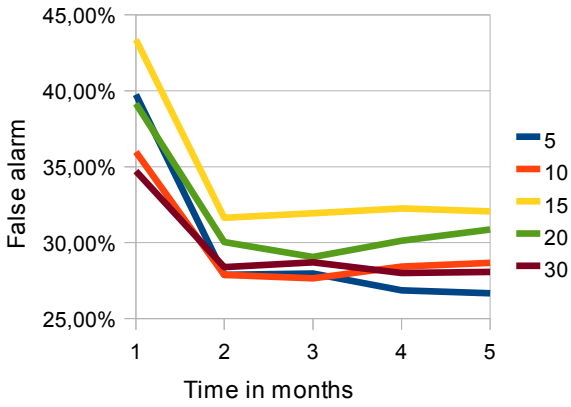


Figure 5.3.4.4: False alarms

Prediction range has only a limited effect on detection rates. The bursty nature of data causes many predictions to be made in a small interval, which eliminates the advantage of a larger range. This is amplified by the fact that these results were produced using zero variance in the copying of training data. Large ranges may be more beneficial when variances are higher.

For a low false alarm, the range should be either very short, or very large. A short range contains fewer transitions, making it less likely that a probability will exceed the threshold when there is no activity. Large ranges include such a large amount of transitions that the effects of individual transitions are cancelled out, also decreasing probabilities in times of inactivity.

Threshold

A prediction with a probability higher than the threshold is considered to be "true". Different threshold settings have an impact accuracy, as is illustrated below.

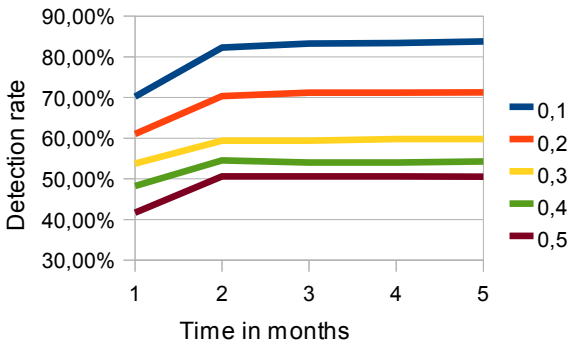


Figure 5.3.4.5: Detection rates

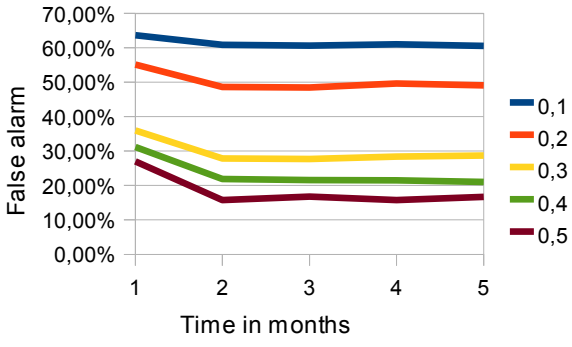


Figure 5.3.4.6: False alarms

A lower threshold increases the detection rate because more predictions will be able to exceed a low threshold. The opposite is true for false alarms. Considering more predictions increases the number of faulty considerations.

Section5.4 Impact analysis of sub-techniques

Our technique combines features from both Markov chains and nearest neighbor classification. This section will give an insight into the individual performance of these techniques and also the influences they have in the performance of transition probability. The implementation of these techniques, however, are adaptations of the original techniques so the presented results do not reflect "pure" Markov chains and nearest neighbor classification.

Markov chains are implemented as transition probability without the incorporation of state similarities and the possibility of learning from similar situations. Nearest neighbor is implemented as transition probability with the exclusion of the point distribution system as described in section 3.3. Initial points are assigned based on state similarity and the number of times a state has occurred. Distribution is skipped and the final processing phase produces a prediction based purely on the points as they are assigned in the initialization phase.

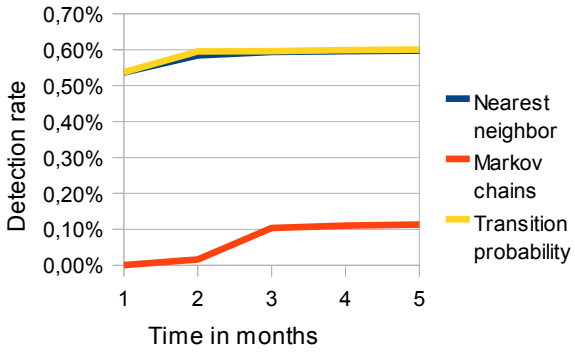


Figure 5.4.1: Detection rates

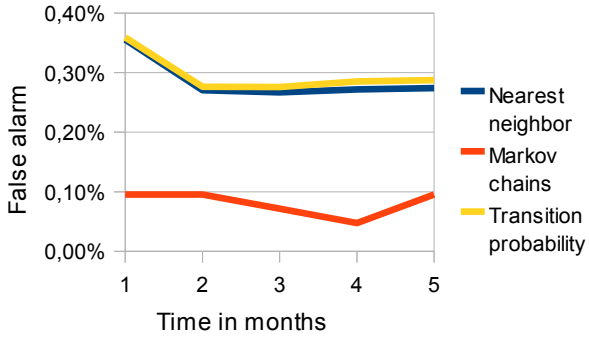


Figure 5.4.3: False alarms

Figures 5.4.1 and 5.4.2 show the detection rates and false alarms for each of the techniques. Remarkable about these results is that nearest neighbor classification has a very similar prediction accuracy to our own technique. This sub-technique seems to be the driving force behind transition probability. Markov chains are clearly struggling with the large amount of different states. The inability to learn from similar states makes it difficult to see patterns in user behavior.

As depicted in figure 5.4.3, the total prediction latency of transition probability is evenly divided between its two sub-techniques. Figure 5.4.4 shows the memory usage per technique. The similarities in these usages indicate that memory usage is independent of the learning technique. Memory is primarily used to store states and transitions.

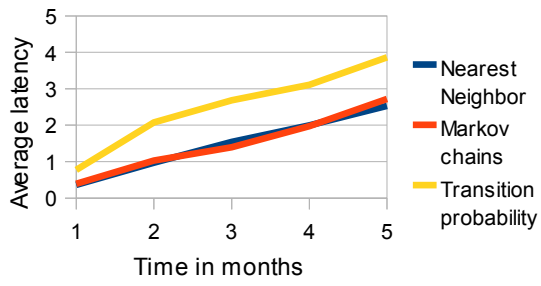


Figure 5.4.3: Average latencies

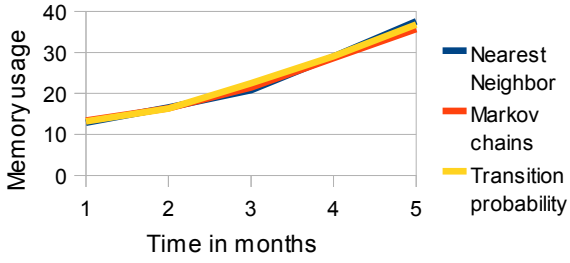


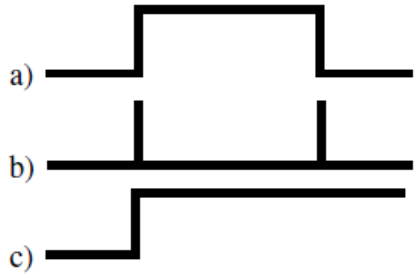
Figure 5.4.4: Memory usages

Section5.5 Comparison with other techniques

A paper called “Accurate activity recognition in a home setting”[ref] describes the application of two different learning techniques to the same data set that was used in the evaluation of transition probability. Hidden Markov models (HMM) and conditional random fields (CRF) are evaluated on their accuracy in recognizing activities. Both techniques learn which activity belongs to a set of sensor data, at a certain time. Time is divided into 60 second time slices and an activity as predicted for each of these slices.

Accuracy is defined using two measures; time slice accuracy, and class accuracy. Time slice accuracy represents the percentage of correctly classified time slices. This is a subtle difference from class accuracy where the average percentage of classified time slices per activity, or class, is captured. Measurements were acquired using a cross-validation method. One full day of sensor is used for testing and the remaining days are used for training. This process is repeated for each day and accuracies are averaged over all days.
An impact analysis on accuracy is performed based on different sensor data representations.

- **Raw (a)** representation leaves the sensor data unaltered.
- **Change point (b)** representation gives a 1 to time slices where the sensor reading changes.
- **Last (c)** representation gives a 1 only the sensor that changed last and 0 in every other case.



Section5.5.1 Accuracy comparison

In order to make a comparison with transition probability, a comparable measure of accuracy is required. Some adaptations are made to make the system perform predictions based on time slices. After adding cross-validation functionality, it is possible to define a time slice accuracy similar to the one used for HMM and CRF. At each time slice, each activity is either occurring or not. The accuracy of the prediction for this slice depends on the probability of the occurrence of this activity. If the activity does occur, the probability of occurrence is added and if the activity does not occur, the probability of non occurrence is added. This results in a summation of probabilities over all time slices per activity. Average accuracy is obtained by dividing the sum of probabilities by the number of predictions.C

Activity	Time slice accuracy
Prepare Dinner	96,77%
Go To Bed	83,94%
Get Drink	98,46%
Use Toilet	89,46%
Prepare Breakfast	98,05%
Take Shower	97,96%
Leave House	68,37%

Combining all results gives the following results:

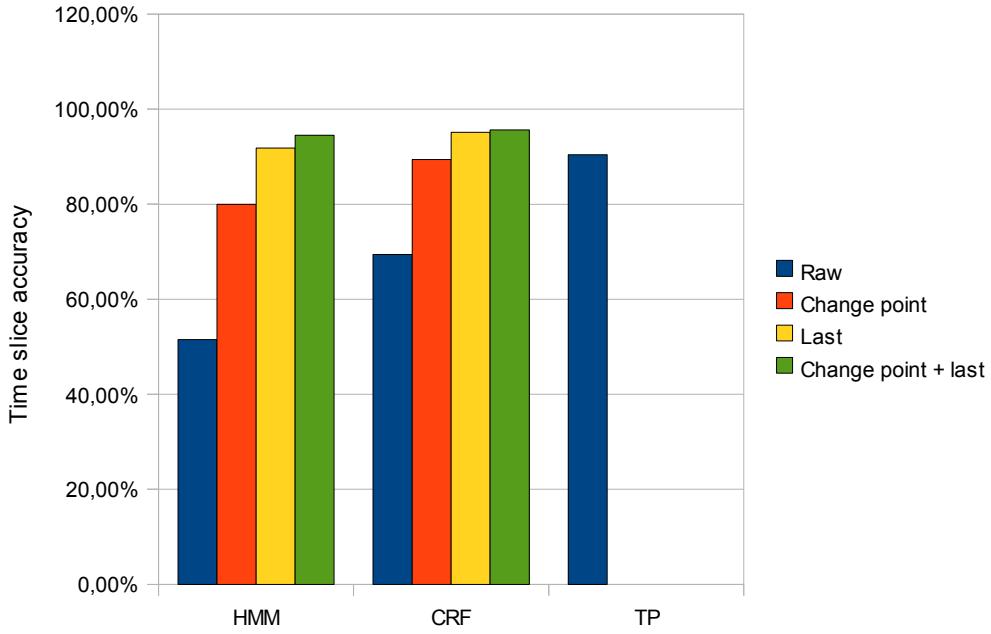


Figure 5.5.1.1: Time slice accuracies for HMM,CRF and TP

Transition probability uses a raw data representation exclusively because other representations are simply not available. Accuracies from the other techniques suggest that a different data representation may significantly improve performance. Therefore, future work on transition probability may include a performance evaluation of these data representations.

Section5.5.2 Accuracy function comparison

What is remarkable about the comparison of these different techniques is that the accuracy function is very different from the one originally used for the performance analysis of transition probability. This difference can be explained based on optimal predictions in the perception of both accuracy functions.

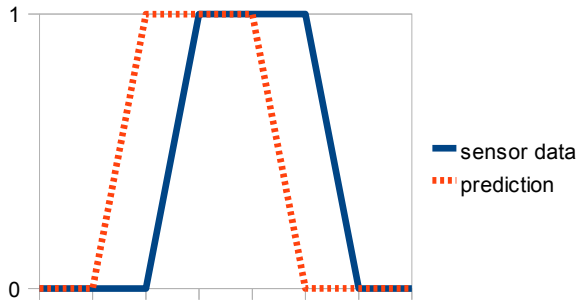


Figure 5.4.2.1: Optimal prediction accuracy

Figure 5.4.2.1 illustrates a prediction with an optimal failure-, and detection rates. Predicted sensor data is equal to the actual sensor data only shifted in time. Predictions are always one step ahead of actual user behaviour. This way, every change in sensor data is always predicted beforehand, yielding a perfect detection rate. No failures occur because every predicted change occurs moments later. Illustrated below are the optimal predictions from the point of view of accuracy function used for comparison.

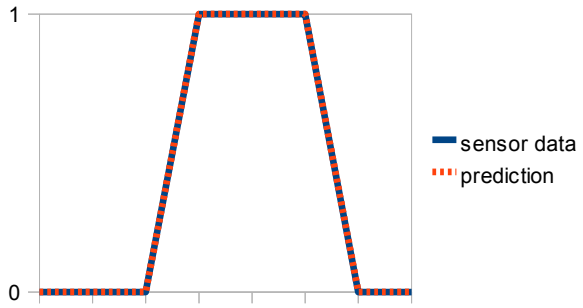


Figure 5.4.2.2: Optimal classification accuracy

With these predictions, every time slice is matched exactly, yielding a perfect accuracy. If the same predictions were to be evaluated by the original function however, the result would suggest that these predictions are not so accurate. false alarms would remain the same but changes in sensor data would never be predicted beforehand because of the exact match up.
These observation lead to the conclusion that the original accuracy function is more based on predicting future behaviour whereas the function used for comparison is more based on classifying current behaviour.

Section5.5.3 Computation time and memory usage

Experiments with activity recognition using hidden Markov models and conditional random fields, as described in [18], were performed in Matlab. The source code of these experiments was made available so that all presented results can be reproduced. This collection of Matlab code will allow us to measure both memory usage and latency so that they can be compared to the measurements performed for transition probability. Accurate measurements will be provided by the build in Matlab profiler tools and gathered on the same computer system used for earlier measurements to ensure consistency and comparability.

Table 5.4.3 gives an overview of computation times and memory usages per technique based on one month of training. Computation time represents the total amount of time needed to process all the training data. In the case of transition probability, this is the sum of the latencies of all predictions within this month.

Technique	Computation time	Memory usage
Transition probability	7,87 seconds	13,2 Mb
Hidden Markov Models	6,31 seconds	4,1 Mb
Conditional random fields	>10 hours	28,1 Mb

Table 5.4.3 : Computation times and memory usages

Section5.5.4 Conclusion

Differences in the definition of accuracy reveal the subtle difference between the goals of traditional learning techniques and those of transition probability. Where HMM and CRF focus on classifying the current situation, transition probability is more aimed towards predicting the future situation. Because the future is derived from the present, classification of the present also plays an import role in the prediction process. This is why transition probability exhibits a comparable classification accuracy to that of HMM and CRF.

Even compared to HMM, transition probability requires relatively small amounts of computation time to process its training data. Total computation time is slightly higher but the advantage compared to HMM is that these computations are not performed all at once, as one undividable block. Computation time is divided into bursts of approximately 2 milliseconds, one burst for each individual prediction. The 7,87 seconds of processing required in total can therefore be spread out over the entire month. Memory usages are relatively low overall and do not put any serious limitations on modern hardware.

Chapter6 Conclusion

This thesis proposed transition probability, a machine learning technique designed with the purpose of saving energy and increasing user comfort in a home environment. Achieving these goals requires learning user habits and predicting user behaviour. If these habits are known, energy could be saved by turning off devices left on unintentionally, and user comfort could be increased by automating the user's interaction with devices. Sensors placed in the home record the usage of appliances and devices. Hidden in this recorded data are patterns, or habits, which are sought after by the learning algorithm. Predictions are made about the reoccurance of these patterns, directly based on sensor data and without the need for a preceding training step, as is common in machine learning.

A performance evaluation expressed the ability to learn user habits in a numerical value. Measurements on the inner workings of the system reveal accuracy, memory usage, and prediction latency. Conditions, both internal and external, were closer examined to uncover the impacts of these conditions on system performance. Such an examination also provided a detailed insight in the hidden characteristics of the algorithm.

The expression of performance into a numerical value makes it possible to objectively compare this technique to other machine learning techniques. Transition probability was compared to hidden Markov models and conditional random fields. Both of these other techniques were able to reach a higher classification accuracy than transition probability. Comparison also revealed possible improvements on the current system. These possibilities, and others, are described in the future work section.

Section6.1 Future work

Transition probability is the product of a relatively short development process. Boundaries were set on the scope of the project to ensure feasibility. This left some alternatives to certain design choices unexplored. The following paragraphs contain a brief overview of these alternatives and the impact they are expected to have on performance.

Section6.1.1 Relevance distribution over time

The current implementation of transition probability features a linear relevance distribution function that provides a simple and highly predictable distribution of influence over the input states. Other possibilities include Gaussian functions and fuzzy quantifiers. Both with their own additional parameters and settings. Experimentation with different functions and different settings are likely to lead to higher detection rates and lower false alarms.

Relevance distribution has so far been interpreted as an influence distribution based on the time of day. Transitions in the morning are considered to be important for predictions made early in the day. This concept of matching useful transitions to predictions could be extended to incorporate influence based on the day of the week. Predictions made on a Monday could benefit from the assignment of more influence to transitions on other Mondays. Or alternatively, transitions that also occurred on a weekday instead of a day in the weekend. Further extensions of this concept could be based on matching months, or even seasons, to each other.

Section6.1.2 Data representation

Comparing transition probability with techniques described in [18]gave some clues as to where possible optimizations may lay. Techniques were evaluated using different representations of the same data set. A raw representation leaves the data unchanged which means that sensor readings are fed directly into the system. This kind of input has so far been the only one used in the evaluation of transition probability. However, both hidden Markov models as conditional random fields scored their lowest accuracy scores using unaltered sensor data. Significantly higher scores were achieved using other representation and it is not unlikely that the same will hold for transition probability.

Section6.1.3 State duration

Some states occur more often than others. States that occur often are more likely to occur in the future and our technique takes this into account when making a prediction. However, there is another aspect to states that may give some clues about future occurrences. A state not only occurs a certain amount of times, it also occurs for a certain period of time. This state duration gives another measure for dominance of a state in the habits of a user. Our technique does not currently incorporate this aspect and including it could have a positive effect on prediction accuracy.

Bibliography

1: Hani Hagras,Victor Callaghan, Martin Colley, Graham Clarke,Anthony Pounds-Cornish, and Hakan Duman, Creating anAmbient-IntelligenceEnvironment UsingEmbedded Agents, 2004
2: Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson, Activity Recognition in the Home Using Simpleand Ubiquitous Sensors, 2004
3: Michael C. Mozer, The Neural Network House:An Environment that Adapts to its Inhabitants, 1998
4: Sayal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin O. Heierman III, Tze-Yun Lin, The role of prediction algorithms in the MavHome smart home architecture,
5: Parisa Rashidi, Diane J. Cook, Keeping the resident in the loop: adapting the smart home to the user,
6: , Go-Green project, , http://gogreen-project.nl/
7: S.Michalski,J.Carbonall and T.Mitchell, Machine learning: An artificial intelligence approach,
8: Jianchao Han, Learning Fuzzy Association Rules andAssociative Classification Rules,
9: Anupam Joshi,Narendran Ramakrishman,Elias N. Houstis, and John R. Rice, On Neurobiological, Neuro-Fuzzy, MachineLearning, and Statistical PatternRecognition Techniques, 1997
10: Nils J. Nilsson, Introduction To Machine Learning, 1998
11: Charles Elkan, Nearest Neighbor Classification, 2011
12: Chris Stauffer, Learning Patterns of Activity Using Real-Time Tracking, 2000
13: David R. Musser and Gor V.Nishanov, A Fast Generic Sequence Matching Algorithm, 2001
14: Zoubin Ghahramani , Unsupervised Learning, 2004
15: S. S. Intille, K. Larson, J. S. Beaudin, J. Nawyn, E. Munguia Tapia, and P. Kaushik, A living laboratory for the design and evaluation of ubiquitous computing interfaces,
16: , , , http://www.bwired.nl/
17: Tim van Kasteren, Athanasios Noulas, Gwenn Englebienne and Ben Kröse, Accurate activity recognition in a home setting,
18: Liming Chen, Chris Nugent, Jit Biswas, Jesse Hoey, Activity Recognition in Pervasive Intelligent Environments, 2010