Zhiphone: A Phone that Learns Context and User Preferences

Jing Michelle Liu and Raphael Hoffmann

CSE567 Project Report

December 14, 2005

Abstract

Users of a mobile phone are often required to adapt its alarm type to context. While ringing tones may be obtrusive during meetings, a vibration alarm is barely noticeable on a bus. We propose setting the alarm type automatically, based on context-information obtained from a variety of low-cost sensors. Our multi-sensor board captures audio, acceleration, compass readings, temperature, humidity, barometric pressure, and ambient light. Based on raw sensor data, we build five soft sensors representing context properties that are relevant for setting a suitable alarm type. In particular, we determine if the user keeps the phone close to the body, is involved in a conversation, is exposed to high background noise, is outdoors, or is physically active. Although a decision for the alarm type can be made based on the context detected, users generally have different preferences as to what alarm type to use in a given context. We therefore propose learning user preferences through reinforcement, by associating reactions like taking or hanging up an incoming call with rewards and punishments. Our experimental results show that we can accurately detect context, and that context and user preference learning can execute in real-time on an off-the-shelf mobile device.

Contents

Chapter 1

Introduction

Users of mobile phones generally prefer different alarm types for different situations. For example, while attending a meeting, studying in a library, or visiting a movie theater, a user may choose to redirect incoming calls to a voice mailbox or be notified by vibration rather than an obtrusive ringing tone. In contrast, when the user is walking on a busy street with high background noise, she may prefer a loud ringing tone that she is able to hear. If her cell phone is located close to the body and she is not walking fast, vibration could be an option, too. However, she won't be able to notice the vibration alarm, when she is running. Although currently available mobile phones are equipped with a variety of alarm types, users are required to switch the alarm setting manually. Having to remember to continuously adapt the alarm setting to different contexts imposes a cognitive load on users. Furthermore, forgetting to adjust the alarm type can be disturbing, such as an interruption by a loud ringing tone in the middle of a meeting. In our work, we attempt to create a context-aware mobile phone, Zhiphone, which adjusts the alarm type automatically based on information about the user's activity or her environment. Although context-aware mobile phones have been proposed in research already, to the best of our knowledge, no phone on the market at the time of writing follows this approach.

We believe that one reason why context-aware mobile phones have not appeared on the market, is that due to the diversity of users the *one-size-fits-all* approach to adapting the alarm type to context fails. Users generally have different preferences as how to be notified of an incoming call in a given situation. For example, some users like using their phones on a bus, while others prefer more privacy. We propose to learn these user preferences through reinforcement, by observing a user's reaction to an incoming call. For example, if a user hangs up without taking the call, we infer that the selected alarm type may have been inappropriate in this situation.

Our objective in this work is to show that we can accurately detect relevant context properties, and that we can learn context and user preferences

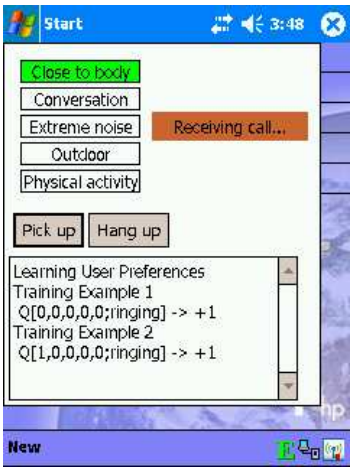

Figure 1.1: **User interface of the Zhiphone application.** The background color of the five boxes in the upper right corner show current outputs of the soft sensors. To their right, a box informs the user about incoming calls. A user can respond to an incoming call by clicking on either the *pick up* or *hang up* button. The generated training example for the reinforcement learner is displayed in the console at the bottom.

on a mobile device in real-time.

At the hardware side, we use an off-the-shelf HP iPAQ 4700 handheld device to simulate a mobile phone. It is connected via USB to a custom-built multi-sensor board that includes 7 sensors and an Atmel ATMega 128L microprocessor running at 7.3728MHz. The available sensors are shown in table C.1. For a more detailed description, refer to appendix C.

Based on raw data provided by the sensors, we created five *soft sensors* that describe properties about a user's activity and her current environment. In particular, we determine if her phone is located close to her body, she is involved in a conversation, she is exposed to loud background noise, she is outdoors, or she is physically active. We track this context information in real-time on the iPAQ and and output the system's believed state on the screen, as shown in figure 1.1. A background thread simulates phone calls at random time intervals and causes alarms. When an incoming call is received, the user can click on either the *pick up* or the *hang up* button. Based on the user's choice and the current setting of the soft sensors, a new training example for the reinforcement learner is generated and displayed on the screen.

In the next chapter, we discuss related work. In chapter 3, we explain our approach in detail. Experimental results are shown in chapter 4. Finally, chapter 5 gives a brief conclusion and discusses the direction of our future work.

Chapter 2

Related Work

In a work that is most closely related to ours, Siewiorek et. al. [8] propose Sensay, a context-aware mobile phone. Sensay's architecture is based on five functional components: the sensor box, the sensor module, the decision module, the action module, and the phone module. The sensor box collects sensor data, the software-based sensor module queries that data, the decision module determines the phone's state, the action module sets that state and the phone module provides access to the mobile phone operating system and user interface. From the raw sensor data, a series of threshold analysis tests are run and in consequence the user's state is determined. SenSay introduces four states: uninterruptible, idle, active and normal. Sensay applies only basic learning techniques to distinguish context. Moreover, there is no learning from user preferences.

Horvitz et.al. [5] propose Bayesphone, a system that applies offline learning and reasoning with user models to predict whether users will attend meetings on their calendar and the cost of being interrupted by incoming calls should a meeting be attended. While the models are precomputed offline, the inferred policies can be cached on a mobile phone.

Horvitz et.al. [4, 3] have also studied a computer user's interruptability from computer activity or contextual information. As features they consider a user's current and recent history of computer activity, meeting status, location, time of day, and whether a conversation is detected. In a training phase, users are intermittently asked to assess their perceived interruptability.

Fogarty et.al. [1] study the interruptability of programmers. As features they use low-level event logs in a development environment. Instead of using manually labeled training data, they measure the response time of users to interruptions.

Chapter 3

Architecture

3.1 Overview

Figure 3.1 depicts the overall architecture of our system. It consists of three main components: the feature extraction module, the supervised learning module, and the reinforcement learning module. The feature extraction module collects raw sensor data from the multi-sensor board and extracts features like frequency components or variance. The supervised learning module uses the pre-processed features and computes the outputs of our five soft sensors. It should be noted that supervised learning does not take place on the iPAQ - it is performed offline using annotated user traces, and only the learned classifiers are used on the iPAQ. The outputs of the soft sensors combined with a user reaction to an incoming call is used as an input to the reinforcement learning module. It maintains a knowledge representation of user preferences and decides about the alarm type.

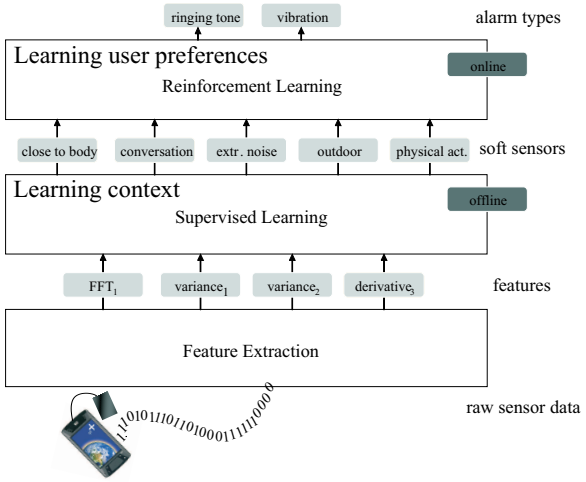

Figure 3.1: **Architecture overview showing data flows and modules.**

Our architectural overview shows data flowing from the multi-sensor board via feature extraction and soft sensor classification up to the reinforcement learner which finally sets the alarm type. To better explain our motivation for various design decisions, however, we decided to present the three components of our system in reverse order.

3.2 Learning User Preferences

Today's mobile phones are already equipped with a variety of alarm types, which can be customized by the user. While some people prefer loud ringing tones, others prefer more subtle and less obtrusive ones. Almost all phones on the market support vibration as an alarm type, and some phones even support notifying the user by flashy lights. Instead of setting an alarm type, a user may also configure a voice mailbox that takes every call directly, or a user may set up call forwarding to another line. There are a wide variety of options available how mobile phones can react to user calls, but since alarm or other reaction types are not the focus of our research, we decided to focus on two basic ones instead.

Ringing Tone We assume a ringing tone that is well audible in loud environments like public transportation systems or restaurants. The tone should not be annoying, but its volume should be inappropriate in libraries, movie theaters, or meetings.

Vibration The vibration alarm should be noticeable when the phone is being carried around close to the body, e.g. in a pocket, and the user is not involved in physical activities. It should only make a small amount of noise such that it can be used in quiet environments like libraries.

With these defined alarm types in mind, we empirically searched for situations for which we could specify a clear preference for one alarm type over the other. The situations we identified, as well as our personal alarm preferences are shown in table 3.1.

Our first observation is that there exist several situations, in which the two subject's preferred alarm settings are not the same. Raphael generally prefers vibration over ringing tone as long as he is able to notice that, whereas Michelle generally prefers ringing tone as long as the environment does not prohibit the use of such. To accommodate for different user preferences, we decided to let the mobile phone automatically adapt to its user through Reinforcement Learning. In the following section we will give an introduction to Reinforcement Learning and finally show how we apply the technique to automatically adapt alarm settings to user preferences.

Situation	Michelle	Raphael
User is involved in a conversation	ringing tone	vibration
User is physically active, e.g. on a treadmill	ringing tone	ringing tone
User puts phone on desk	ringing tone	ringing tone
User leaves phone in jacket and puts jacket away	ringing tone	ringing tone
User rides bus	ringing tone	vibration
User walks outdoors	ringing tone	vibration
User is in library	vibration	vibration
User is studying in silent office environment	vibration	vibration
User listens to loud music, e.g. in disco	ringing tone	ringing tone

Table 3.1: **Preferred alarm types for two subjects in various situations.**

3.2.1 Reinforcement Learning

Reinforcement Learning [6, 7] is a way of programming an agent through reward and punishment. Unlike in a supervised setting, the agent is not being given examples of what action to take in a given state, but it must learn through trial-and-error what actions will yield a high long-run measure of reward.

Let's put this idea into a more formal setting. Our model consists of a discrete set of environment states, $\mathcal{S}$, a discrete set of actions, $\mathcal{A}$, and a set of numeric rewards, $\mathcal{R}$. Let $R(s, a)$ denote the reward received when in state s and executing action a, and let $T(s, a, s')$ give the probability of reaching state s' when in state s and executing action a. Our goal is to learn a policy π, mapping states to actions, with a high long-run measure of reward. Typical measures are the h-step finite-horizon model which gives the expected reward for the next h steps, or the infinite-horizon discounted model, which geometrically discounts rewards received in the future by a discount factor γ. The discount factor implies that immediate rewards have a higher value than rewards received far in the future. From now on, we assume an infinite-horizon discounted model.

To solve a Reinforcement Learning problem, one typical approach is to compute the optimal value of every state $V^*(s)$. $V^*(s)$ is the expected infinite discounted sum of reward that the agent gains when it starts in state s and executes the optimal policy. $V^*(s)$ can be obtained by solving the Bellman equations

$$V^*(s) = \max_a (R(s, a) + \gamma \sum_{s \in \mathcal{S}} T(s, a, s') V^*(s'), \quad \forall s \in \mathcal{S}.$$

An agent's optimal action in state s is then given by the action a that maximizes the right term of this equation. Unfortunately, the transition function T is often unknown and thus the equations cannot be solved directly. How-

ever, it can be shown that updates of the form

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

approximate a function $Q^*(s,a)$ such that $V^*(s) = \max_a Q^*(s,a)$. In the update formula, it is assumed that s' is sampled from the distribution $T(s,a,s')$, and r is sampled with mean $R(s,a)$ and bounded variance, and the learning rate α is decreased slowly. $Q^*(s,a)$ gives the expected discounted reward of taking action a in state s and acting optimally afterwards.

We can update $Q(s,a)$ every time we receive an immediate reward, and as our next action a we can choose the one that maximizes $Q(s,a)$. Unfortunately, this strategy does not let us learn an optimal policy, because we may not know about other actions yielding even more reward than the one that currently maximizes $Q(s,a)$. Therefore, we need to trade-off exploitation and exploration. We choose a randomized strategy that by default takes the action with the best estimated expected reward, but with some probability p, it selects a random action.

3.2.2 Applying Reinforcement Learning

We now discuss how the problem of automatically adapting the alarm type to user preferences can be cast as a Reinforcement Learning problem.

It is straightforward to define the state and action spaces. As a state space $\mathcal{S}$ we choose a discrete set of tuples (s_a, s_s), where s_a indicates the current alarm setting and s_s is drawn from a set of contextual situations. This set of contextual situations is a generalization of the situations given in table 3.1 and will be discussed in the next section. Our action space $\mathcal{A}$ only consists of two actions, switching to ringing tone and switching to vibration alarm.

The main challenge in defining our problem using the Reinforcement Learning framework, however, lies in the definition of a system of rewards and punishments. We need to be able to obtain some feedback from the user that lets us conclude whether our previous actions were satisfactory. The approach we chose is based on the immediate reaction of a user to an incoming call. The reactions of the user and their associated reward or punishment are shown in table 3.2.

User reaction	Reward/Punishment
User takes call	small positive reward
User hangs up, without taking call	large negative reward
User does not respond to call	small negative reward

Table 3.2: **User reaction to a incoming call and its associated reward or punishment.**

An obvious question that now arises is – how do we justify our choice of reward or punishment for the given situations? If the user takes an incoming call, we can infer that the user was able to notice the phone's alarm and that he is able to take the call. In the second case, the user hangs up without taking the call. We conclude that the user is not able to take the call, and the fact that he hangs up indicates to us that the alarm type may be inappropriate or obtrusive in the given situation. Finally, if a user does not respond to a call, it may be the case that he was unable to notice the alarm.

Setting the exact reward parameters is, of course, a difficult task. The parameters affect the behavior of the Reinforcement Learner, whose usefulness can only be determined by conducting experiments on real subjects. Unfortunately, real-world data is expensive to obtain, since all data examples should be drawn from the same user and it is only possible to obtain a single training example per phone call. Furthermore, the data is generally very noisy, since users are often inconsistent in their reaction to phone calls. For example, they may sometimes have a conversation interrupted by a phone call and sometimes not. Adjusting the reward parameters of the Reinforcement Learner can be compared to model selection in a supervised learning.

To approximate the Q function we use a multi-layer perceptron and the Backpropagation learning algorithm. The Backpropagation algorithm is especially suitable for our setting, since every update requires only a constant and modest execution time and can thus be run in real-time on the iPAQ.

3.3 Learning Context

In section 3.2, and in particular in table 3.1, we presented a variety of situations that can be used in setting a phone's alarm. In this section, we intend to generalize these situations to tuples of high-level context properties which we can learn in a supervised setting.

We identified five properties that can be used to describe the context. Our selection of properties was guided by the motivation to distinguish the aforementioned situations and by the requirement that these properties are learnable from the sensor data provided by the multi-sensor board.

Close To Body (CB) Determines if the phone is located close to the body. If it is close to the body, we expect small variations in accelerometer or compass data.

Conversation (CN) Determines if the user is involved in a conversation. If the user is involved in a conversation, we expect unique patterns in the time and frequency domain data of the microphone.

Extreme Noise (EN) Determines if the user is exposed to high noise levels. If exposed to high noise levels, we expect high amplitudes in microphone data.

Outdoor (OU) Determines if the user is outdoors. If outdoors, we expect different settings for temperature, humidity, ambient light, background noise.

Physical Activity (PH) Determines if the user is physically active. If physically active, we expect large variations in accelerometer data.

Table 3.3 shows our expected matching between the aforementioned situations and our defined properties.

Situation	CB	CN	EN	OU	PH
User is involved in a conversation	$*$	T	N	$*$	$*$
User is physically active, e.g. on a treadmill	$*$	$*$	$*$	$*$	T
User puts phone on desk	F	$*$	$*$	F	$*$
User leaves phone in jacket and puts it away	F	$*$	$*$	$*$	$*$
User rides bus	T	$*$	T	T	F
User walks outdoors	T	$*$	$*$	T	F
User is in library	$*$	F	F	F	F
User is studying in silent office environment	$*$	F	F	F	F
User listens to loud music, e.g. in disco	T	$*$	T	F	$*$

Table 3.3: **Properties of contextual situations.** Entries marked with T or F indicate that the corresponding properties are present or not present, respectively. For entries marked with $*$, the value of the property is irrelevant.

In the following sections, we first give an introduction to the supervised learning framework that we employed, and afterwards show how it is applied to learn our set of context properties.

3.3.1 Support Vector Learning

In its simplest form, the Support Vector Machine [10] is a linear classification algorithm. For two linearly separable sets of data in feature space, there generally exist an infinite number of separating hyperplanes. The Support Vector Machine, however, yields the unique solution that maximizes the distance between the closest training points of each class and the separating hyperplane.

In formal notation, if $(\mathbf{x}_i, y_i),\ 1 \leq i \leq m$ denotes a set of training example with $y_i \in \{-1, +1\}$, then our goal is to find the separating hyperplane, given by the normal vector $\mathbf{w}$ and offset from the origin b, that solves the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} \quad & y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) \geq 1 \quad \text{for all } 1 \leq i \leq m. \end{aligned}$$

The constraints require all training examples to be classified correctly and with positive distance from the hyperplane. It can then easily be shown that it's optimum the objective function gives the distance between the closest training points and the hyperplane.

Unfortunately, due to outliers a separating hyperplane may not exist. Therefore, we need to relax our conditions and allow some training points to lie too close to the hyperplane or be even misclassified. Rewriting our optimization problem, we get[1]

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}||\mathbf{w}||^2 + \tfrac{C}{2}\textstyle\sum_{i=1}^{m} \xi_i^2 \\ \text{subject to} \quad & y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) \geq 1 - \xi_i \quad \text{for all } 1 \leq i \leq m \\ & \xi_i > 0 \quad \text{for all } 1 \leq i \leq m. \end{aligned}$$

In this formulation, the slack variables ξ equal 0 for examples that are classified correctly and lie far enough from the hyperplane, otherwise ξ equals the squared error given by $(1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))^2$. C is a regularization parameter that allows to trade off the number of misclassifications and the smoothness of the solution. The smoothness is relevant when we consider more complicated solutions than hyperplanes. Support Vector Machines offer an elegant technique to find nonlinear solutions by the introduction of kernel functions. It is common to use the Gaussian kernel, in which case the Support Vector Machine algorithm yields solutions that have round or twisting shapes.

3.3.2 Applying Support Vector Learning

We train a Support Vector Machine classifier for each of the five context properties using labeled training data. The features contained in the dataset are not raw sensor inputs, but features extracted in a preprocessing step that is discussed in section 3.4. Since the output of the Support Vector Machine classifiers are used as inputs to higher-level learning algorithms, we will from now on refer to the classifiers as *soft sensors*.

One observation is that we do not need to run the Support Vector Machine algorithm on the iPAQ. We can train the classifiers offline using labeled

[1] We give the L2-C-SVM formulation introduced by Suykens [9, 2]. It is less commonly used than the original L1-C-SVM formulation by Vapnik [10], but has been shown to give similar classification results and is slightly more efficient with our selected optimization solver.

training data and later run the pre-trained classifiers on the iPAQ. The importance of this becomes clear when we consider the fact that the running time of the Support Vector Machine algorithm is generally unpredictable[2].

The running time of the pre-trained classifiers, however, only depends linearly on the number of used support vectors and this number is upper bounded by the number of training examples and is fixed.

3.4 Feature Extraction

The raw sensor outputs from the multi-sensor board are generally not suitable as an immediate input to a soft sensor classifier. Since the classifiers only take features from a given point in time as inputs, they are not able to capture temporal relationships. For example, the actual value of a sensor reading may be irrelevant to the classification task, but only the fact that this value has been increasing for a period of time or, say, the fact that it has been highly variant for some time. The sensor readings may also be very noisy, which requires us to smooth the signal before using the sensor value. We identified five feature extractors that are useful in learning our defined set of soft sensor classifiers.

Running Mean The mean of the sensor readings over a sliding time window.

Running Variance The variance of the sensor readings over a sliding time window.

Derivative The difference quotient between two sensor readings.

Exponential Smoothing Weighted sum of previous extracted feature and new sensor reading.

Fast Fourier Transform The frequeny representation of the sensor readings.

Feature selection was performed manually based on the observed correlation between raw sensor readings and annotations. Figure 3.2 shows eight raw sensor readings and annotations for a 46-minute recorded trace.

For all but the exponential smoothing extractor, we need to specify the size of a sliding window. This size, of course, has an impact on the extracted

[2]To the best of our knowledge, the running times of all commonly used quadratic programming solvers is highly sensitive to the distribution of the data. Small variations in the data can have a significant impact on the running time.

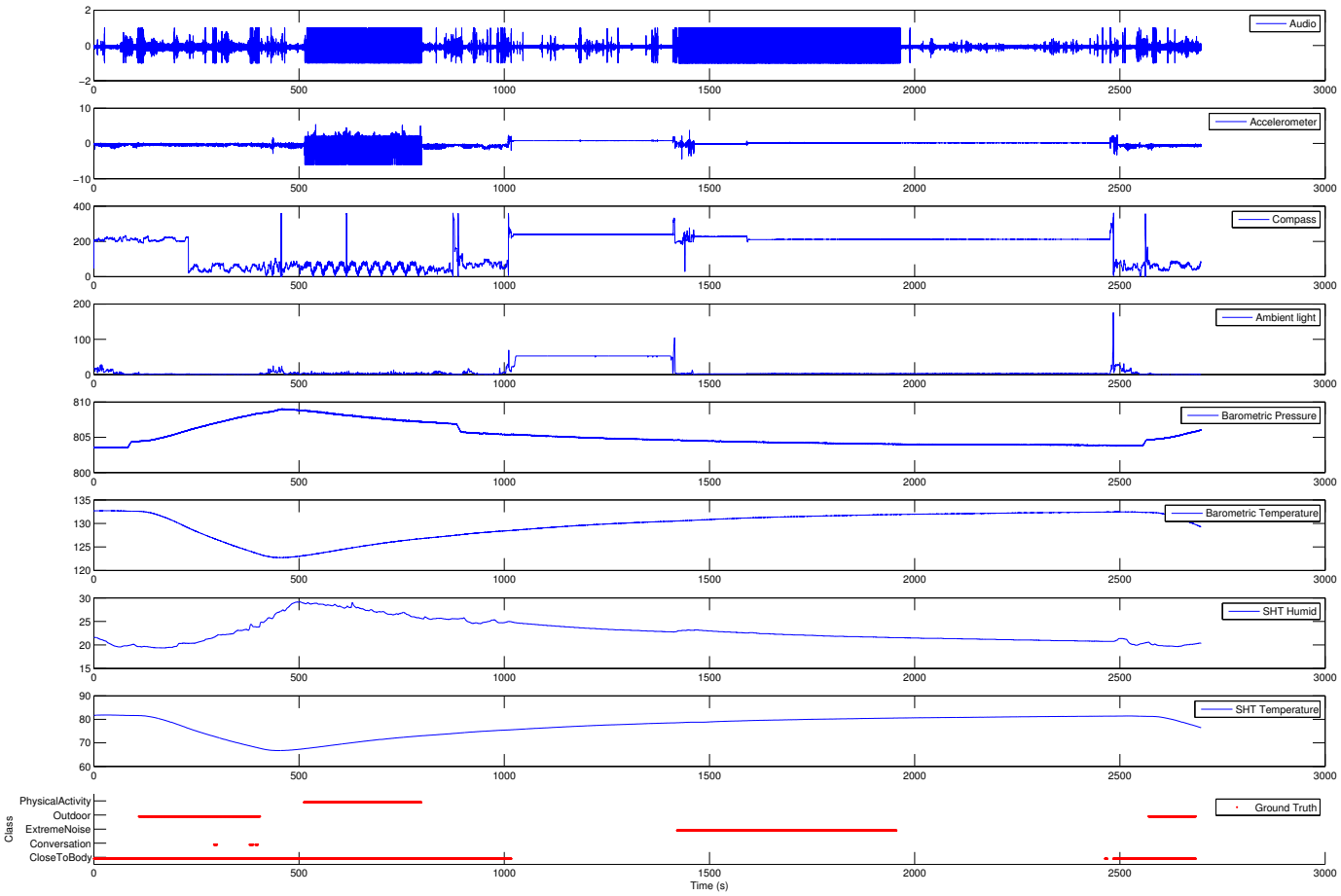

Figure 3.2: **Annotated user trace with eight raw sensor outputs.** For physical activity annotations, the acceloremeter signal (sum of three accelorometer readings, one for each direction) shows high variance and also the amplitude of the audio signal remains highly variant. For outdoor annotations, we see activity in the barometric pressure, barometric temperature, SHT humidity, and SHT temperature signals. All four sensors are adapting to the outdoor environment, however, their signal change is very slow. We conclude that not only their absolute values, but also their derivatives may be useful features for indoor/outdoor discrimination. For the extreme noise annotation, we see high amplitudes in the audio signal. For close to body annotations, we see minor variations in the accelorometer, compass, and ambient light signals.

features. In our experiments we have found that we would ideally like to have different window sizes for different sensors. For example, the close to body sensor is more accurate when the underlying features are computed on a relatively long sliding window of one minute, since a user may simply hold still for a couple of seconds. On the other hand, the conversation sensor performs better, if we compute the Fast Fourier Transform on a shorter window of 10 seconds on the audio signal. This keeps the computational overhead smaller and well recognizes frequencies that can be typically observed in voice conversations. We therefore use window sizes of 10 second and 1 minute lengths.

It is also useful to compose several feature extractors. For example, we may wish to compute the FFT of the audio signal and afterwards smooth the frequency components over time. Also, the derivative values are only useful when the underlying signal exhibits enough smoothness. This can be achieved by first applying a running mean or exponential smoothing filter before computing the derivative. Since there are a large number of possible and useful feature extractor compositions, we developed each extractor as a pluggable component filter, where each filter can have multiple inputs and multiple outputs. In addition to flexibility in composition, the components were developed with regard to efficiency and an economical usage of buffering.

Since feature extraction must run on the iPAQ in real-time, we require low and predictable execution times for each feature extractor. All presented feature extractors require only a constant number of CPU instructions.

Chapter 4

Design of Experiments

Our goal in this paper is to show that automatically adapting a mobile phone's alarm type to context and user preferences is technically feasible. In particular, we would like to show that

1. the context can be accurately described by building the proposed soft sensor classifiers,

2. all components of the system (feature extraction, soft sensor classification, and reinforcement learning) can execute on the iPAQ in real-time.

In the following sections, we show these points through experimental evaluation.

4.1 Accuracy of Learning Context

To evaluate the accuracy of our soft sensors we collected two user traces in and around Paul G. Allen Center, each roughly one hour in length. Each trace contains several minutes of annotations of each of the five context properties that our soft sensors should recognize. An overview of the traces is given in table 4.1.

	Trace 1	Trace 2	Total
Close To Body	48 min	31 min	79 min
Conversation	24 min	5 min	29 min
Extreme Noise	4 min	9 min	13 min
Outdoor	6 min	10 min	16 min
Physical Activity	5 min	5 min	10 min
Total recorded time	62 min	55 min	117 min

Table 4.1: **Durations of annotations and traces collected at the Paul G. Allen Center.** Trace 1 and Trace 2 are each roughly one hour in length and each contain several minutes of recordings of all five annotations.

Using one of the two traces as a training and the other as a test set, we trained a Support Vector Machine with Gaussian kernel for each of the annotations. We optimized the algorithm parameters (the regularization parameter C and the Gaussian kernel width σ) by searching on a regular grid in the log space of the parameters, and selecting the combination that yielded the lowest 5-fold cross-validation error on the training set. Table 4.2 shows the average results of choosing Trace1 as a training and Trace2 as a test set, and of choosing Trace2 as a training and Test1 as a test set.

	Precision (in %)	Recall (in %)	Accuracy (in %)
Close To Body	88.058	87.647	82.799
Conversation	77.879	32.247	81.678
Extreme Noise	81.569	88.547	96.511
Outdoor	68.478	61.204	91.874
Physical Activity	94.309	79.293	97.876
Average	82.059	69.788	90.148

Table 4.2: **Precision, Recall, and Accuracy for different Support Vector Machine classifiers.** Each value was computed by 2-fold cross-validation using the Trace1 and Trace2 datasets. Precision is given by (# true positives) / (# true positives + # false positives), recall is given by (# true positives) / (# true positives + # false negatives), and the accuracy is given by (# true positives + # true negatives) / (size of the dataset).

The overall average accuracy was above 90%. We find this number encouraging and believe that even higher accuracies can be attained. For example, we did not perform automatic feature selection, nor did we ignore training examples at annotation boundaries. Furthermore, it is possible to interpret the distance of a test example to a classifiers's decision boundary as a confidence value in the prediction. Using these confidence values, soft sensor predictions can further be smoothed by Kalman filters or a probabilistic graphical model.

4.2 Real-time Performance of all Components

In addition to accurately detecting context, we require real-time performance of feature extraction, soft sensor classification, and reinforcement learning on the iPAQ. By real-time performance, we mean that a new context can be recognized within several seconds.

We begin our analysis by measuring the execution times of our feature extractors on the iPAQ. The results are shown in table 4.3.

Except for the Fast Fourier Transform all feature extractors required far less than a second of execution time.

The execution times for the soft sensors are given in table 4.4. Except for the conversation soft sensor, each soft sensor requires well under a second

Feature Extractor	Time
Mean (window size 20)	0-1 ms
Variance (window size 5500)	15-18 ms
Derivative	0-1 ms
Exponential Smoothing	0-1 ms
Fast Fourier Transform (window size 4096)	1032-1083 ms
Window (window 5500, w/ overlap)	96-109 ms

Table 4.3: **Measured execution times of various feature extractors on the iPAQ.** For each extractor, the number of executed instructions is identical for every run; running time variations are mainly due to simultaneously running threads. The running time of several extractors depends on the used window size. The sizes we selected in this table are actually used in our system to obtain 10 seconds or one minute windows with the multi-sensor board's sensor sampling rates.

for classification. For comparison, we also include the training times of the classifiers on a 1.5GHz Pentium-M laptop.

	Classification	Training
Close To Body	127 ms	4 sec ($\pm$ 2)
Conversation	(memory error)	212 sec ($\pm$ 134)
Extreme Noise	74 ms	3 sec ($\pm$ 2)
Outdoor	132 ms	4 sec ($\pm$ 2)
Physical Activity	12 ms	3 sec ($\pm$ 2)
Average	$\geq$ 86 ms	45 sec ($\pm$ 28)

Table 4.4: **Measured executions of soft sensor classification on the iPAQ and soft sensor training on a laptop.** For training times, the numbers in brackets indicate standard deviation. The classification times are deterministic and did not vary significantly on the iPAQ. The memory error was due to the fact that the Support Vector Machine extracted too many high-dimensional support vectors that could not all fit into the iPAQs modest memory. This problem can be solved by reducing the number of support vectors or by reducing the number of features. The large number of features was due to computing the FFT on the audio signal.

Finally, we measure the execution times of the Reinforcement Learner. We use a 3-layer perceptron to represent the Q-function. Every time the user receives an incoming call, the system queries the perceptron with its current soft sensor states and one alarm type to see the predicted reward. The system chooses the alarm type that gives the highest expected reward. After the user reaction is available, a single training example is generated and the state of the perceptron is updated using a Backpropagation step. Table 4.5 shows that both, the prediction and Backpropagation update, require only a negligible running time.

	Prediction	Backpropagation update
Multi-layer Perceptron	0-1 ms	0-1 ms

Table 4.5: **Measured executions of Reinforcement Learner on the iPAQ.** We represent the Q-function by a 3-layer perceptron with 6 input nodes, 5 hidden nodes, and 1 output node.

We have therefore shown that all components of our system can execute on the iPAQ in real-time.

Chapter 5

Conclusion and Future Work

In this work we have shown that it is technically feasible to automatically adapt the alarm type of a mobile phone to its context and user preferences. In particular, we have shown that we can accurately detect relevant context properties and that all components of the system – feature extraction, soft sensor classification, and reinforcement learning – can execute on an off-the-shelf mobile device in real-time.

While our work focused on the technical feasibility of the approach, more research needs to concentrate on studying real-world experiences of users with the system. Since this was not one of our goals, we did not perform comprehensive user studies. However, we believe that at this point user studies are essential to advancing research in context- and user-aware mobile phones.

A prerequisite to conducting realistic user evaluations is to equip the iPAQ with real phone capability. Since the iPAQ is already able to communicate via 802.11 wireless networks, we would like to use a VoIP library like JVOIPLIB[1]to enable making real phone calls.

In combination with user evaluations, it may then be promising to more carefully explore and evaluate our or alternative Supervised Learning and Reinforcement Learning strategies. For example, we did not evaluate how quickly the Reinforcement Learner converges to the real preferences of a user. This question, however, is essential, especially since multi-layer perceptrons typically require a large number of Backpropagation updates to approximate a function. Unfortunately, it is difficult to obtain training data, since we only get a single training example per incoming call, we can only consider data collected by one user, and finally the collected data is generally very noisy.

[1]http://research.edm.luc.ac.be/jori/jvoiplib/jvoiplib.html

Bibliography

[1] J. Fogarty, A. J. Ko, H. H. Aung, E. Golden, K. P. Tang, and S. E. Hudson. Examining task engagement in sensor-based statistical models of human interruptibility. In *CHI*, pages 331–340, 2005.

[2] T. V. Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. D. Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54(1):5–32, 2004.

[3] E. Horvitz and J. Apacible. Learning and reasoning about interruption. In *ICMI*, pages 20–27, 2003.

[4] E. Horvitz, P. Koch, and J. Apacible. Busybody: creating and fielding personalized models of the cost of interruption. In *CSCW*, pages 507–510, 2004.

[5] E. Horvitz, P. Koch, R. Sarin, J. Apacible, and M. Subramani. Bayesphone: Precomputation of context-sensitive policies for inquiry and action in mobile devices. In *User Modeling*, pages 251–260, 2005.

[6] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res. (JAIR)*, 4:237–285, 1996.

[7] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[8] D. P. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F. L. Wong. Sensay: A context-aware mobile phone. In *ISWC*, pages 248–249, 2003.

[9] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[10] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

Appendix A

Implementation

We developed our system using the Java Development Kit, version 1.3.1. Our system uses several freely available libraries:

- Our implementation of the Support Vector Machine algorithm uses the LibSVM quadratic programming solver[1].
- Our reinforcement learner is an adapted version of PIQLE[2].
- Our system uses the UWAR libraries to communicate with the multi-sensor board.

A.1 Acknowledgements

We would like to thank Brian Ferris for his support in setting up real-time data capture from the multi-sensor board.

[1] http://www.csie.ntu.edu.tw/∼cjlin/libsvm/
[2] http://www.lifl.fr/∼decomite/piqle/piqle.html

Appendix B

Source Code and Instructions

B.1 Source Code

The full source code including required com.intel.research.uwar packages can be downloaded from

`http://www.cs.washington.edu/homes/raphaelh/research/cse567.zip`.

B.2 Prerequisites

To run Zhiphone you will need

- A HP iPAQ 4700, running Microsoft PocketPC
- A UW Multi-Sensor Board connected to the iPAQ via USB
- Microsoft ActiveSync
- Eclipse 3.0 or higher
- Java Development Kit 1.3 or higher
- J9 Java Virtual Machine on the iPAQ

B.3 Installation

Unpack cse567.zip. Before the Zhiphone application can compile in Eclipse, one needs to make certain configurations in Eclipse:

1. Create a new Eclipse project for each of the following directories in cse567.zip:

- `com.intel.research.uwar.patterns`
- `com.intel.research.uwar.io`
- `com.intel.research.uwar.jtracewriter`
- `cse567`
- `org.eclipse.swt`

2. For each checked out module open the Properties window and set the Java compiler to version 1.3 (higher versions are not supported with the J9 JVM that runs on the iPAQ).

3. For com.intel.research.uwar.patterns, an additional Eclipse library is required that is not included in the standard version. I've included that library in the `/libs` directory of the `cse567` module. So, for the `com.intel.research.uwar.patterns` module, load the properties dialog, select the *Java Build Path* tab, select *Libraries*, click *Add external Jar*, and select the `/libs/swt.jar` file from the `cse567` module.

4. Additionally, for each module add certain other modules to the Build Path. (Open Properties, Java Build Path, Projects tab).

   - For `com.intel.research.uwar.patterns`
     add nothing
   - For `com.intel.research.uwar.io`
     add `com.intel.research.uwar.patterns`
   - For `com.intel.research.uwar.jtracewriter`
     add `com.intel.research.uwar.io`
   - For `cse567`
     add `com.intel.research.uwar.io`
     add `com.intel.research.uwar.jtracewriter`

B.4 Running Zhiphone on the iPAQ

To run the application on the iPAQ, do the following steps:

1. Call Eclipse-File-Export.
   Select JAR-File.
   Select the src directory in each of the above listed modules.
   In the last dialog window set the entry point to be the `Main` class.

2. Save JAR file on desktop, e.g. cse567.jar.

3. Create a text file on the desktop, containing exactly the following line

```
255#\j9\bin\j9.exe "-Xbootclasspath:\j9\lib\classes.zip;
\j9\lib\swt.jar;\j9\lib\charconv.zip" -cp "/My Documents
/cse567.jar;" "zhiphone.main.Main" ""
```

4. Copy JAR and text file onto iPAQ using ActiveSync.
5. Rename text file on ipaq to cse567.lnk (one can do that in the active sync window on the desktop)
6. Run by clicking on cse567.lnk on iPAQ.

Appendix C

Multi-Sensor Board

The multi-sensor board is built on a 6-layer PCB that includes an Atmel ATMega 128L microprocessor running at 7.3728MHz. The MSB is equipped with 7 sensors (listed in Table C.1) which are all controlled by the on-board microcontroller as shown in Figure C.1. The MSB has two communication modes: either wired (via a USB or compact flash bridge) or wireless (via a bluetooth). It can collect data on handheld, desktop computers and cell phone. The onboard microprocessor controls all the low level I/O operations necessary to communicate with each sensor. The microprocessor performs the sampling (either polled or streaming) and then relays the data over a hi-speed UART (Universal Synchronous and Asynchronous serial Receiver and Transmitter) to the devices which are connected to receive the serial data stream.

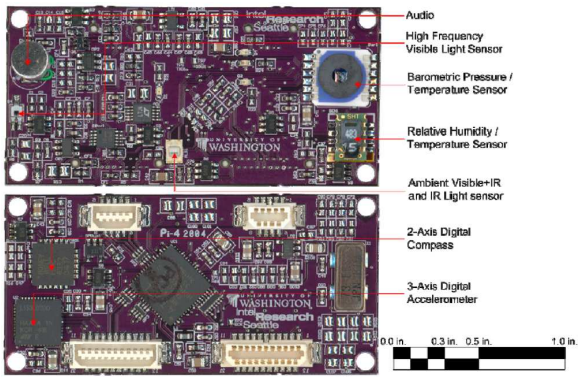

Figure C.1: **Layout of sensors on the MSB.**

Manufacturer	Part No.	Description	Sampling Rate
Panasonic	WM-61A	Electric Microphone	$\sim$ 16000 Hz
Osram	SFH-3410	Visible Light Phototransistor	$\sim$ 550 Hz
STMicro	LIS3L02DS	3-Axis Digital Accelerometer	$\sim$ 550 Hz
Honeywell	HMC6352	2-Axis Digital Compass	30 Hz
Intersema	MS5534AP	Digital Barometer / Temperature	15 Hz
TAOS	TSL2550	Digital Ambient Light	5 Hz
Sensirion	SHT15	Digital Humidity / Temperature	2 Hz

Table C.1: **Specifications of sensors on the MSB.**