Context-Aware Energy Enhancements for Smart Mobile Devices

Brad K. Donohoo, *Student Member, IEEE*, Chris Ohlsen, *Student Member, IEEE*, Sudeep Pasricha, *Member, IEEE,* Yi Xiang, *Student Member, IEEE,* Charles Anderson, *Member, IEEE*

Abstract— Within the past decade, mobile computing has morphed into a principal form of human communication, business, and social interaction. Unfortunately, the energy demands of newer ambient intelligence and collaborative technologies on mobile devices have greatly overwhelmed modern energy storage abilities. This paper proposes several novel techniques that exploit spatiotemporal and device context to predict device wireless data and location interface configurations that can optimize energy consumption in mobile devices. These techniques, which include variants of linear discriminant analysis, linear logistic regression, non-linear logistic regression with neural networks, k-nearest neighbor, and support vector machines are explored and compared on synthetic and user traces from real-world usage studies. The experimental results show that up to 90% successful prediction is possible with neural networks and k-nearest neighbor algorithms, improving upon prediction strategies in prior work by approximately 50%. Further, an average improvement of 24% energy savings is achieved compared to state-of-the-art prior work on energy-efficient location-sensing.

Index Terms— energy optimization, pervasive computing, machine learning

1 Introduction

Mobile phones and other portable devices (tablets, PDA's, and e-readers) are fundamental everyday tools used in business, communication, and social interactions. As newer technologies (e.g. 4G networking, multicore/GPUs) and applications (e.g. 3D gaming, Apple's FaceTime™) gain popularity, the gap between device usage capabilities and battery lifetime continues to increase, much to the annoyance of users who are now becoming more and more reliant on their mobile devices. The growing disparity between functionality and mobile energy storage has been a strong catalyst in recent years to develop software-centric algorithms and strategies for energy optimization [1]-[17]. These software techniques work in tandem with well-known energy optimizations implemented in hardware including CPU DVFS, power/clock gating, and low power mode configurations for device interfaces and chipsets [18]-[21].

The notion of "smart" mobile devices has recently spawned a number of research efforts on developing "smart" energy optimization strategies. Some of these efforts employ strategies that are context-aware including utilization of device, user, spatial, temporal, and application awareness that attempt to dynamically modify or learn optimal device configurations to maximize energy savings with little or negligible impact on user perception and quality of service (QoS) [10][13][17]. This general theme of a *smart* and *context-aware* energy optimization strategy is further explored in this paper, in which a select number of machine learning algorithms are proposed and evaluated for their effectiveness in learning a user's mobile device usage pattern pertaining to spatiotemporal and device contexts, to predict data and location interface configurations. These resulting predictions manage the network interface states allowing for dynamic adaptation to optimal energy configurations, while simultaneously maintaining an acceptable level of user satisfaction. This idea is further motivated by considering the power distributions of the Google Nexus One Android smartphone illustrated in Figure 1 [22]. Even when 3G, WiFi, and GPS interfaces are all enabled and idle, they account for more than 25% of total system power dissipation. Furthermore, when only one of the interfaces is active, the other two idle interfaces still consume a non-negligible amount of power. Our work exploits this fact to save energy more aggressively than the default energy management strategy used

- *B. K. Donohoo is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: bdonohoo@rams.colostate.edu*
- *C. Ohlsen is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: ohlsensc@rams.colostate.edu*
- *S. Pasricha is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: sudeep@colostate.edu*
- *Yi Xiang is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: yix@colostate.edu*
- *Charles Anderson is with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: anderson@cs.colostate.edu*

This article is a significantly extended version of our paper accepted for publication in ACM/IEEE DAC 2012 titled "Exploiting Spatiotemporal and Device Contexts for Energy Efficient Mobile Embedded Systems" with the following major additions: (i) More comprehensive related work in Section 2 explaining how our work is different and novel in comparison with previously published architectures; (ii) Further description of the synthetic user profiles in Section 3.4; (iii) More detailed descriptions of the machine learning algorithms in Section 4, including a list of pros and cons for each algorithm; (iv) Analysis and results for a new machine learning technique based on support vector machines; (v) Further description of our power modeling efforts in Section 5; (vi) A study involving dimensionality reduction using principal component analysis (PCA) in Section 6.3; and (vii) A more accurate analysis of the implementation overhead for each of the algorithms in Section 6.4, in which the algorithms are run on an actual mobile device.

Digital Object Indentifier 10.1109/TMC.2013.94

in a mobile device, by dynamically managing data and location interfaces, e.g., turning off unnecessary interfaces at runtime.

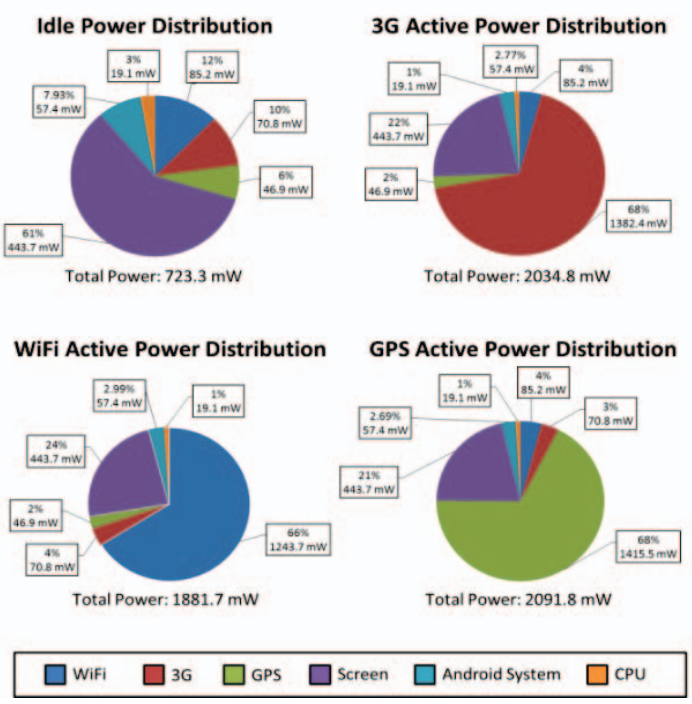


Figure 1: Google Nexus One smartphone power distributions

In this paper, we propose and demonstrate the use of five different classes of machine learning algorithms *(i) linear discriminant analysis, (ii) linear logistic regression, (iii) k-nearest neighbor, (iv) non-linear logistic regression with neural networks, and (v) support vector machines*, towards predicting user data/location usage requirements based on spatiotemporal and device contexts. These strategies are tested on both synthetic and real-world user usage patterns, which demonstrate that high and consistent prediction rates are possible. The proposed techniques are also compared with prior work on device configuration prediction using self-organizing maps [13] and energy-aware location sensing [9], showing an improvement upon these state-of-the-art techniques.

The remainder of this paper is organized as follows. Section 2 reviews several key related works. Section 3 discusses the acquisition and creation of real and synthetic user profiles that are used in our analysis studies. Section 4 provides a brief overview of the machine learning concepts used in the studies. Section 5 describes our device power modeling effort. Section 6 presents the results of our experimental studies. Finally, Section 7 presents our conclusions and ideas for future work.

2 Related Work

A large amount of work has been done in the area of energy optimization for mobile devices over the past decade [37], [38]. Much of the recent work focuses on optimizing energy consumed by the device's wireless interfaces by intelligently selecting the most energy-efficient data interface (e.g. 3G/EDGE, WiFi) [1], [2]. Other work [3]-[8] focuses on energy-efficient location-sensing schemes aiming to reduce high battery drain caused by location interfaces (e.g. WiFi, GPS) by deciding when to enable/disable location interfaces or modify location acquisition frequency. Lee et al. [9] in particular propose a Variable Rate Logging (VRL) mechanism that disables location logging or reduces the GPS logging rate by detecting if the user is standing still or indoors. The authors in [10] propose a context-aware method to determine the minimum set of resources (processors and peripherals) that results in meeting a given level of performance, much like our work. They determine if a user is moving/stationary and indoors/outdoors and control resources using a static lookup table. In contrast, *our work controls resources dynamically by using machine learning algorithms*. Zhuang et al. [11] propose an adaptive location-sensing framework that involves substitution, suppression, piggybacking, and adaptation of an application's location-sensing requests to conserve energy. Their work is directed towards LBAs (location-based applications) and only focuses on location interfaces, while *ours is a system-wide optimization strategy that is capable of saving energy regardless of the foreground application type*.

A substantial amount of research has been dedicated to utilizing machine learning algorithms for the purpose of mobile user context determination. Batyuk et al. [13] extend a traditional self-organizing map to provide a means of handling missing values and then use it to predict mobile phone settings such as screen lock pattern and WiFi enable/disable. Other works attempt to predict the location of mobile users using machine learning algorithms. In [14] the authors propose a model that predicts spatial context through supervised learning, and the authors in [15] take advantage of signal strength and signal quality history data and model user locations using an extreme learning machine algorithm. These works are focused on using user context for device self-configuration and location prediction, *whereas our work is focused on using user context for optimizing energy consumed by data transfer and location interfaces*.

The authors in [28] apply machine learning techniques to energy-efficient sensing, as we do in this work. They group context sensors into three categories according to their energy efficiency, using the more energy-efficient sensors to infer the status of high-energy-consuming sensors so that activating them may not be necessary. The major difference between their work and ours is that they use machine learning techniques for learning the inference models to capture relationships between groups of sensors (e.g., between energy-efficient software sensing and high-energy consuming hardware sensing categories), *whereas in our work, machine learning techniques are used to learn user-specific spatiotemporal and device contexts*.

One of the key motivations for applying pattern recognition and classification algorithms to mobile device usage is the observation that user usage patterns are often mutually-independent, in that each user generally has a unique device usage pattern. The use of pattern recognition then allows for energy optimization algorithms to be fine-tuned for each user, achieving energy savings without perturbing user satisfaction levels. This idea is further confirmed in mobile usage studies [16], which additionally focused on smartphone usage pattern analysis and its implications on mobile network management and device power management. Although their work had a slightly different focus than our work, the key relevant take-away is that the authors demonstrated from a two month real smartphone usage study that *all users have unique device usage patterns*. Many other works [30]-[36] utilize data gathered from groups of real smartphone users to emphasize this same conclusion. Our previous work [17] also found that usage patterns are unique in the way users interact with different apps on their mobile devices. In Section 4 of this study, the real user usage patterns further confirm this claim – all five users had unique usage

patterns in the amount of interaction as well as when and where the interactions most often took place.

3 User Interaction Studies

In this work we focus on exploiting learning algorithms to discover opportunities for energy saving in mobile systems through the dynamic adaptation of data transfer and location network interface configurations without any explicit user input. Consequently, we enable energy-performance tradeoffs that are unique to each user by efficiently enabling/disabling their device's network interfaces. In order to compare the relative effectiveness of our different learning algorithms (described in Section 4) at predicting a user's data/location usage requirements, five real user usage profiles for five different Android smartphones (HTC myTouch 3G, Google Nexus One, Motorola Droid X, HTC G2, and Samsung Intercept) were collected over a one week period with a custom Context Logger application. The application logged user context data on external storage, which was acquired at the end of the one week session and used in our algorithm analysis.

Table 1: Recorded data attributes

Context	Attribute	*Type*
Temporal	Day of week	*Discrete*
	Time of day	*Discrete*
Spatial	Latitude	*Continuous*
	Longitude	*Continuous*
	GPS Satellite Count	*Discrete*
	WiFi RSSI	*Discrete*
	Number of WiFi APs Available	*Discrete*
	3G Network Signal Strength	*Discrete*
	Device Moving	*Logical*
	Ambient Light	*Discrete*
Device	Call State	*Discrete*
	Battery Level	*Discrete*
	Battery Status	*Discrete*
	CPU Utilization	*Continuous*
	Context Switches	*Discrete*
	Processes Created	*Discrete*
	Processes Running	*Discrete*
	Processes Blocked	*Discrete*
	Screen On	*Logical*
Targets	Data Needed	*Logical*
	Coarse Location Needed	*Logical*
	Fine Location Needed	*Logical*

3.1 Context Logger

We created a custom *Context Logger* application that ran in the background as an Android service and gathered both spatiotemporal and device usage attributes at a one minute interval. Table 1 lists the attributes recorded by the logger and used for algorithm analysis and indicates whether the attribute was a continuous variable (floating point), discrete variable (integer), or logical variable (true/false). The *GPS Satellites* attribute is used as an indirect correlation to GPS signal strength and *WiFi RSSI* is a measure of WiFi signal strength. In addition to more common device attributes such as *Battery Level* and *CPU Utilization*, we gathered several uncommon OS attributes: *Context Switches*, *Processes Created*, *Processes Running*, and *Processes Blocked*. We hoped to aid prediction by using these as inputs to the machine learning algorithms. The three target variables (*Data Needed*, *Coarse Location Needed*, and *Fine Location Needed*) were obtained by examining the requested Android permissions of all of the device's current running foreground applications and services. The *Device Moving* attribute was determined by using the accelerometer sensor and a metric for movement that is the sum of the unbiased variance of X, Y, and Z acceleration [7], given as:

$$Var(m_1 \dots m_n) = \frac{\sum_{i=1}^{N} m_i^2 - \frac{1}{N}(\sum_{i-1}^{N} m_i)^2}{N-1} \quad (1)$$

$$Metric = Var(x_1 \dots x_n) + Var(y_1 \dots y_n) + Var(z_1 \dots z_n) \quad (2)$$

3.2 Data Preparation

As mentioned earlier, GPS location coordinates were recorded along with the other attributes at one minute intervals. The Android SDK GPS location data returns the user's longitude and latitude coordinates in decimal degrees as reported by the onboard GPS chipset [26]. Although exact accuracy is dependent on the actual GPS hardware, the returned values were truncated to a given precision and each longitude and latitude coordinate pair was mapped to a unique location identifier. The truncated location resolution generalized the number of unique locations in which a user spends his/her time, given that for example, a user's home may consist of several different samples of different longitude and latitude pairs. In addition, temporal conditions can be applied to further reduce the number of unique locations (e.g. disregard locations where user spent less than x minutes). Figure 2 shows the effect of truncation and application of temporal conditions for a real user and how the primary locations where the user spent most of his/her time are revealed. In the figure, the leftmost pie chart shows that, without truncation, there are too many unique locations to effectively use for prediction (each color is a different unique location). In the pie charts to the right, the precision of the GPS coordinates is reduced and more significant locations can be seen. For our study we used a location precision of 4 decimal places, as it offers a good balance between effectiveness and accuracy.

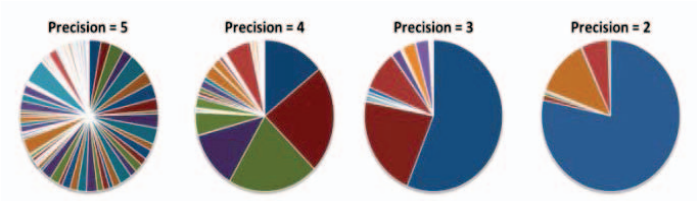


Figure 2: Unique locations identified for varying GPS precisions

The desired data/location interface configurations were partitioned into eight different states based on the desired target variables (*Data Required*, *Coarse Location Required*, and *Fine Location Required*). Table 2 maps the logical values of the three target variables to a state. The states define the device's current required resources. *Efficiently predicting one of these 8 states using temporal, spatial, and device context input variables in Table 1 may ultimately allow opportunistic shutdown of location/wireless radios*. If all interfaces are enabled, they would consume a significant amount of energy in their idle states without this dynamic control. Some might wonder about the worth of predicting users' needs, considering that applications and services request what they need at runtime. However, it is important to realize that applications are rarely designed with energy efficiency in mind – software developers are generally

more concerned about performance and functionality rather than optimizing for energy efficiency.

Table 2: Interface configuration states

State	Data Required	Coarse Location Required	Fine Location Required
1	No	No	No
2	No	No	Yes
3	No	Yes	No
4	No	Yes	Yes
5	Yes	No	No
6	Yes	No	Yes
7	Yes	Yes	No
8	Yes	Yes	Yes

3.3 Real User Profile Analysis

We distributed the Context Logger application to five different mobile device users, and logged their context data over the course of one week. Figure 3 demonstrates the relative state distributions as described in Table 2 for the five different real users. As can be seen, states 2 – 4 are never realized as data was always required when location was required. In addition, the distributions highlight that *each user had a considerably different usage pattern, motivating the need for user-specific adaptation framework to most effectively save energy*. User 1 can be categorized as a minimal user, where the user rarely utilized their phone with only brief periods of interaction, while users 2 and 5 used their phones rather frequently and for longer periods of time. Users 3 and 4 can be categorized as moderate users, primarily utilizing their devices for only certain times of the day.

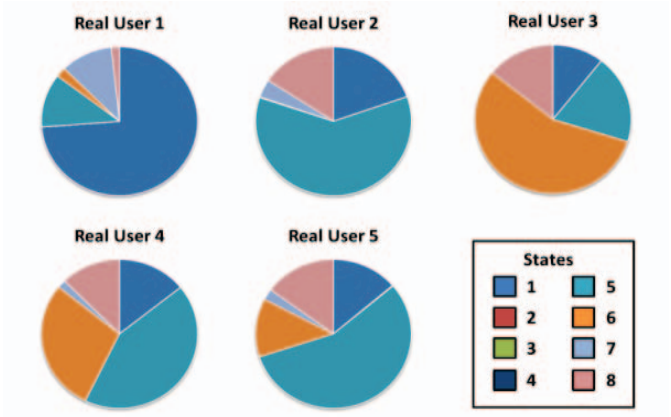


Figure 3: Real user state distributions

4 Learning Algorithms: Overview

The notion of searching for patterns and regularities in data is the fundamental concept in the field of pattern recognition and data classification. *Machine learning* is often focused on the development and application of computer algorithms in this field [23]. In this work we use machine learning algorithms to learn and predict both the spatiotemporal and device contexts of a user, ultimately providing energy savings through efficient control of their device's network configuration. In other words, given a set of input contextual cues, the algorithms will exploit learned user context to dynamically classify the cues into a system state that precisely governs how data and location interfaces are utilized. The goal is to achieve a state classification (Table 2) that saves energy while maintaining user satisfaction by using the recorded data attributes in Table 1. An overview of the basic underlying concepts and application of the machine learning algorithms used in this study is briefly discussed below. The pros and cons of each algorithm are summarized in Table 3.

4.1 Linear Discriminant Analysis

Linear discriminant analysis (LDA) makes use of a Bayesian approach for classification in which parameters are considered as random variables of a prior distribution. This concept is fundamentally different from data-driven linear and non-linear discriminant analyses in which what is learned is a function that maps or separates samples to a class. Bayesian estimation and the application of LDA is also known as *generative modeling*, in that what is learned is a probabilistic model of the samples from each class. By considering parameters as random variables of a prior distribution one can make use of prior known information. For example knowing that a mean μ is very likely to be between a and b, the probability can be determined in such a way that the bulk of the density lies between a and b [24]. Given a prior probability distribution for a particular state classification and a state likelihood, Bayes' theorem (equation 3) can be invoked to get an inferred posterior probability to derive a state prediction (S_k) for a new observed sample of input attributes x_n using a *maximum a posteriori* (*MAP*; equation 4) [24]:

$$p(S_k|x_n) = \frac{p(S_k)p(x_n|S_k)}{p(x)} \tag{3}$$

$$\underset{c}{\operatorname{argmax}}\, p(S_k|x_n) \tag{4}$$

LDA is applicable to a wide range of classification problems of both univariate or multivariate input spaces and binary- or multi-class classification. A number of statistical probability distribution functions can be applied, but the most common is the *Gaussian* or *Normal* distribution (which we use in our study) as shown in equation 5 below.

$$\boldsymbol{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{5}$$

In our work, we obtain an appropriate state classification using LDA by calculating means μ and variances σ for the probability distributions of each state using the input data. We then use the means and variances to calculate discriminant functions for each possible state classification using the Gaussian distribution. Finally, we choose the state corresponding to the discriminant function that outputs the highest value for prediction.

4.2 Linear Logistic Regression

Similar to LDA, linear logistic regression (LLR) is a technique used to derive a linear model that directly predicts $p(S_k|x_n)$, however it does this by determining linear boundaries that maximize the likelihood of the data from a set of state classification samples instead of invoking Bayes' theorem and generating probabilistic models from priori information. LLR expresses $p(S_k|x_n)$ directly by requiring all linear function values to be between 0 and 1 and that they all sum to 1 for any value of x_n, as shown in equation 6:

$$p(C_k|x_n) = \frac{f(x_n,\ \beta_k)}{\sum_{m=1}^{K} f(x_n,\ B_m)} \tag{6}$$

with

$$f(x_n, \beta_k) = e^{\beta_k x_n} \tag{7}$$

With LDA, Bayes' theorem, state priors, and state probability models were used to infer the state classification posterior probabilities, which were then used to discriminate between the different states for a given sample of input attributes x_n. In contrast, LLR solves for the linear weight parameters, β_k, directly using gradients to maximize the data likelihood. This is done by enumerating the likelihood function, $L(\beta)$, using a 1-of-K coding scheme for the target variables, as shown in equation 8 [23], in which every value of $t_{n,k} \in \{0,1\}$ and each row only contains a single '1'. The class variable transformations are known as indicator variables and are used in the exponents of the likelihood function to select the correct terms for each sample x_n.

$$S = \{k_1, k_2, k_3, \dots, k_n\} \downarrow \begin{pmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,k} \\ t_{2,1} & t_{2,2} & \dots & t_{2,k} \\ \vdots & & & \\ t_{n,1} & t_{n,2} & \dots & t_{n,k} \end{pmatrix} \quad (8)$$

$$L(\beta) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(S_k|x_n)^{t_{n,k}} \quad (9)$$

In order to find the β that maximizes the data likelihood, we transform product of products to a sum of sums using the natural logarithm to simplify the gradient calculation with respect to β. Since equation 9 is non-linear, we use an iterative method known as *scaled conjugate gradient* (SCG) [25] (discussed briefly in the following subsection) to solve for the gradient of the log likelihood, $LL(\beta)$, and obtain the respective β weights.

$$LL(\beta) = \sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \log p(S_k|x_n) \quad (10)$$

The iterations repeat until the log likelihood $LL(\beta)$ appears to be at a maximum, then we can plug the updated β weights into equation 6 to obtain the final state probabilities. The resulting state prediction is the state with the highest probability.

4.2.1 Scaled Conjugate Gradient

The scaled conjugate gradient (SCG) algorithm was originally proposed by Moller in 1997 as a method for efficiently training feed-forward neural networks [25]. Simple *gradient descent* algorithms use a fixed step size δ when following a gradient. However, when fixed it is difficult to choose an optimal value for δ, which may result in slow convergence times. Instead, it is better to perform a series of one-dimensional iterative searches known as line searches in the direction of the gradient to choose δ in each iteration. Although using line searches to choose δ is better than using a fixed step size, there are a few problems associated with the resulting gradient descent algorithm. For example, because the gradient descent directions interfere, a minimization along the gradient in one direction may spoil past minimizations in other directions. This problem is solved using *conjugate gradient* methods, which compute non-interfering conjugate directions. Figure 4 shows an example of the two algorithms, gradient descent (red line) and conjugate gradient (green line) beginning at point x_0, then moving along the gradient to find a minimum at point x.

Standard conjugate gradient algorithms still use line searches along the conjugate directions to determine step size. However, there are several drawbacks to doing line searches that can be detrimental to the performance of the algorithm, such as the error calculations involved with each iteration. Scaled conjugate gradient (SCG) algorithms substitute the line search by scaling the step size δ depending on success in error reduction and goodness of a quadratic approximation of the likelihood [25].

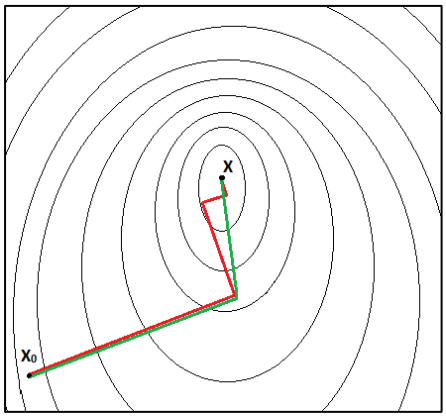


Figure 4: Comparison of gradient descent (red) and conjugate gradient (green) algorithms

4.3 Non-linear Logistic Regression with Neural Networks

Neural network models, also known as *Artificial Neural Networks*, are inspired by the way the human brain is believed to function. Many of the normal basic everyday information processing requirements handled by the brain, for example sensory processing, cognition, and learning, surpass any capable computing system out there today. Although a human brain is quite different than today's computing hardware, it is believed that the basic concepts still apply in that there is a computational unit, known as a *neuron*, and connections to memory stored in *synapses*. The main difference being that the human brain consists of billions of these simple parallel processing units, (neurons) which are interconnected in a massive multi-layered distributive network of synapses and neurons [24].

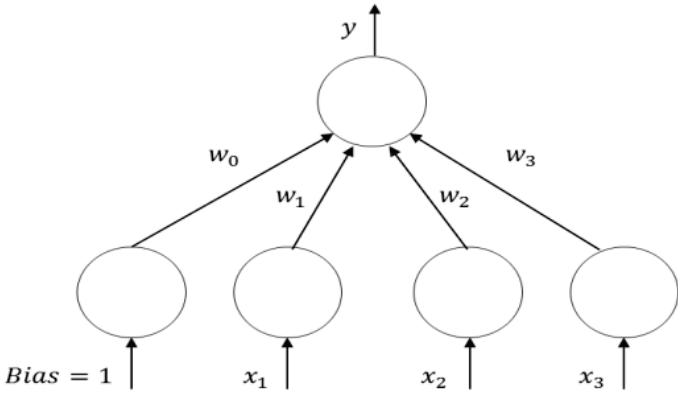


Figure 5: Neural network perceptron model

In machine learning these concepts are modeled by what is referred to as a *perceptron*, which is the basic processing element that is connected with other perceptrons through weighted connections, as illustrated in Figure 5. The output of a perceptron is simply a weighted sum of its inputs including a weighted bias, as shown in equation 11.

$$y = \sum_{i=1}^{n} w_i x_i + w_0 \quad (11)$$

To compute the output y given a sample x_i, backpropagation using the gradient with respect to the weights is performed using a training dataset to find the weight parameters, w_i, that minimize the mean squared error between the neural network outputs, y_i, and the target outputs, t_i. By default, the neural network consists of a hyperplane (for multiple perceptrons) that can be used as a linear discriminant to linearly separate the classes. To improve prediction accuracy, we make it non-linear, by applying a sigmoidal or hyperbolic tangent to hidden unit layer perceptrons (not shown in Figure 5), as denoted in equation 12. This allows for non-linear boundaries with the output of the neural network being linear in the weights, but non-linear in the inputs.

$$y_i' = sigmoid(y_i) = \frac{1}{1 + \exp(-\boldsymbol{w^T x})} \tag{12}$$

For classification with a neural network (non-linear logistic regression), the number of parallel output perceptrons is kept equal to the number of classes in our work. Therefore, our neural network implementation has eight outputs – one for each interface configuration state shown in Table 2. The output from each perceptron, y_i, is then sent to post processing as in equation 13 to determine the respective state prediction by taking the maximum of the post-processed outputs:

$$C_i \; if \; y_i = \max_i \frac{\exp(y_i)}{\sum_i \exp(y_i)} \tag{13}$$

One of the biggest criticisms about the use of neural networks is the time required for training. Although this can be a major issue if using a simple gradient descent approach, newer training techniques, such as the scaled conjugate gradient (SCG) [25], described in Section 3.2.1, can greatly minimize the time required for training. SCG was used for training the neural networks in this study.

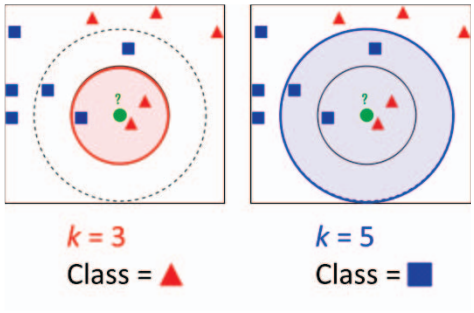


Figure 6: K-nearest neighbor example

4.4 K-Nearest Neighbor

The k-nearest neighbor (KNN) algorithm is a fairly simple non-parametric unsupervised approach for the data classification problem. A key assumption of non-parametric estimation is that similar inputs have similar outputs [23]. In KNN, new samples are classified by assigning them the class that is the most common among the k closest samples in the attribute space. This method requires some form of distance measure for which Euclidean distance is typically used. The Euclidean distance between two points a and b, each containing i attributes, is defined in equation 14.

$$d(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{14}$$

Consider the following example that demonstrates the working of this approach. Figure 6 shows a sample data set, characterized as blue squares and red triangles. The green circle is a new sample that needs to be classified as either a blue square or as a red triangle. If k = 3 (represented by the smaller inner circle with radius 3), the new sample is classified as a red triangle because there are more red triangles within the considered area. Similarly, if k = 5 the new sample is classified as a blue square.

In our implementation, each point is a vector of the input attributes in Table 1, and the calculation of Euclidean distance is performed as a vector operation. This allows us to obtain a configuration state prediction using the gathered context information.

4.5 Support Vector Machines

Support Vector Machines (SVMs) have become quite popular in recent years. SVM is a non-probabilistic binary linear classifier, which constructs a line (for data with dimensionality $D = 2$) or a hyperplane (for data with dimensionality $D > 2$) to separate each given input into one of two possible classes. Examples closest to the separating line or hyperplane are the *support vectors*, and the goal of the SVM is to orientate the line or hyperplane to be as far as possible from the closest members of both classes (largest perpendicular distance). This is known as the *maximum-margin hyperplane*.

Often, the classes to discriminate may not be linearly separable in the original problem's dimensional space. This is especially true in our case – the attribute space for our gathered context data is quite large, and can vary greatly depending on the user. To correct this, SVMs map the original dimensional space to a much higher-dimensional space by using nonlinear *kernel functions*. This allows the algorithm to fit the maximum-margin hyperplane in the high-dimensional feature space, while still allowing it to be nonlinear in the original input space. We use a *radial basis function* (RBF) for the kernel in our SVM implementation.

In an SVM, given K input attributes and n training samples, with class labels -1 or 1, each training sample is mapped to a training data point in a K-dimensional space. Then, the objective is to learn a hyperplane $y = w \cdot x - b$, where w is a K-dimensional vector and x is a K-dimensional data point. The optimization problem is as follows. We have

$$\min_{w,\delta} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \delta_i \right\} \tag{15}$$

subject to

$$c_i(\boldsymbol{w} \cdot x_i - b) \geq 1 - \delta_i \tag{16}$$

where C is a constant value, δ_i is the degree of misclassification of the i-th training data point, and c_i is the category label of x_i.

As stated previously, SVMs are binary classifiers. Thus, in order to classify data into more than two classes, the problem must be broken down into multiple binary classification problems. A common method for such decomposition is to build binary classifiers which distinguish between (i) one of the labels and the rest (one-versus-all) or (ii) every pair of classes (one-versus-one). Classification of new instances for the one-versus-all approach is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class. For the one-versus-one approach, classification is done by a max-wins voting strategy, in which every classifier assigns the

instance to one of the two classes, then the vote for the assigned class is increased by one vote, and ultimately the class with the most votes determines the instance classification. Our SVM implementation uses one-versus-one approach when predicting an interface configuration state for each sample of input attribute.

Table 3 briefly summarizes the pros and cons of the five machine learning techniques that we adapt in our work, based on our implementation experience.

Table 3: Pros and cons of machine learning algorithms

Algorithm	Pros	Cons
LDA	Unbiased; very fast; easy implementation	Not good choice for classes with very different underlying covariance matrices
LLR	Easy to interpret; can model synergistic relationships	Complicated; sensitive to outliers and hard to extrapolate
NN	Can derive meaning from complicated or imprecise data; can handle large number of features; fast	Training can be difficult; slow training time; examples must be selected carefully; not probabilistic, hard to implement
KNN	Simple to understand; data can be scalars or multidimensional vectors; does not make assumptions about underlying data distributions	Slow; non-parametric; all training data is needed during testing phase; all training examples are saved in memory – storage problem
SVM	Can deal with very high dimensional data; good generalization performance; potential for feature selection and outlier detection; general high accuracy	Need to select a good kernel function; high memory/cpu time requirements; slow training time for one-versus-one approach; parameter selection is data dependent

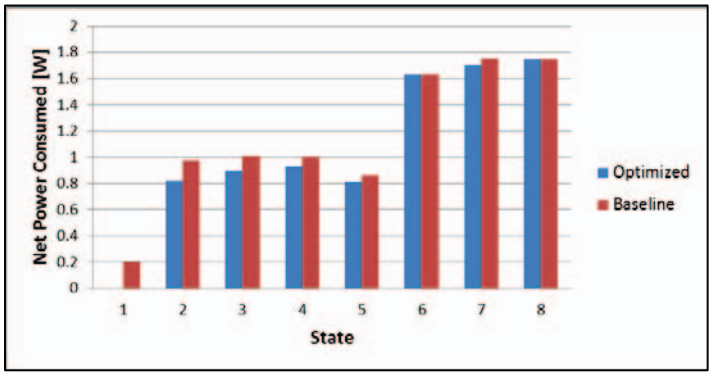


Figure 7: Power consumption of configuration states

5 Device Power Modeling

In order to quantify the energy-effectiveness of using machine learning algorithms to predict energy-optimal device states, power analysis was performed on real Android based smartphones, with the goal of creating power models for the data and location interfaces. We use a variant of Android OS 2.3.3, (Gingerbread) and the Android SDK Revision 11 as our baseline OS. We built our power estimation models using real power measurements, by instrumenting the contact between the smartphone and the battery, and measuring current using the Monsoon Solutions power monitor [27]. The monitor connects to a PC running the Monsoon Solutions power tool software that allows real-time current and power measurements over time. We manually enabled the data/location interfaces one by one and gathered power traces for each interface in their active and idle states. The power traces from the Monsoon Power Tool were then used to obtain average power consumption measurements for each interface. These average power consumption measurements allowed us to determine average power consumption for each of the eight configuration states, shown in Figure 7. The figure shows two measurements for each state – *Optimized* and *Baseline*. *Optimized* shows the average power consumed with the unnecessary interfaces disabled, while *Baseline* shows the average power consumed with the unnecessary interfaces enabled and idle. Intuitively, there is no optimized bar for state 1 because no power is consumed when none of the interfaces are enabled. We use the Baseline measurements for comparison in the experiments in the next section.

6 Experimental Results

In addition to testing our energy saving techniques on five real user profiles, a set of synthetic user profiles were also created for five different idealized and generalized models of average user usage patterns including the following: *(i) 8 – 5 Business Worker*, *(ii) College Student*, *(iii) Social Teenager*, *(iv) Stay At Home Parent*, and *(v) Busy Traveler*. Both an indoor/outdoor location timeline and an interface state profile were created for each synthetic user, as shown in Figure 8. Given the difficulty of generating realistic device system data, such as context switches, CPU utilization, and processes created, only a subset of the attribute space was considered. The remaining attributes were based on both the desired state and/or location. For example, if a user was at an outdoor location, larger GPS satellite values and weak WiFi RSSI values were used as opposed to when the user was indoors. We created the synthetic profiles ourselves by modeling what we considered typical behavior of each stereotype. For instance, we envisioned the 8-5 Business Worker waking up at 6:30 a.m., driving to work at 7:30 a.m., arriving at his/her desk at 8:30 a.m., working until noon then taking a lunch break, etc. The interface configuration states and locations in the charts attempt to capture this behavior. Recall that the locations are just unique location identifiers (integers). In the case of the Busy Traveler, the red line indicating location is constantly changing because the Busy Traveler constantly moving to new locations. The dips in the red line are present because indoor locations are numbered lower than outdoor locations – indicating that the Busy Traveler spent most of his/her time driving outside, but made occasional stops at indoor locations for food, rest, or relief.

6.1 Prediction Accuracy Analysis

Recall that the input attributes for the learning algorithms come from the gathered spatiotemporal and device context data (Table 1), and the predicted output is one of the 8 interface configuration states (Table 2). To evaluate the prediction accuracy of the different algorithms, the data for each user was randomly partitioned into training and test sets using an 80/20 partitioning scheme. The algorithms were then trained on the training data and evaluated on the test data. When being evaluated on the test data, each algorithm's predictions were compared with the target variables from the actual user data to determine the prediction accuracy. This was repeated five times for each implementation and the net prediction accuracy is presented in Figure 9 for the real users.

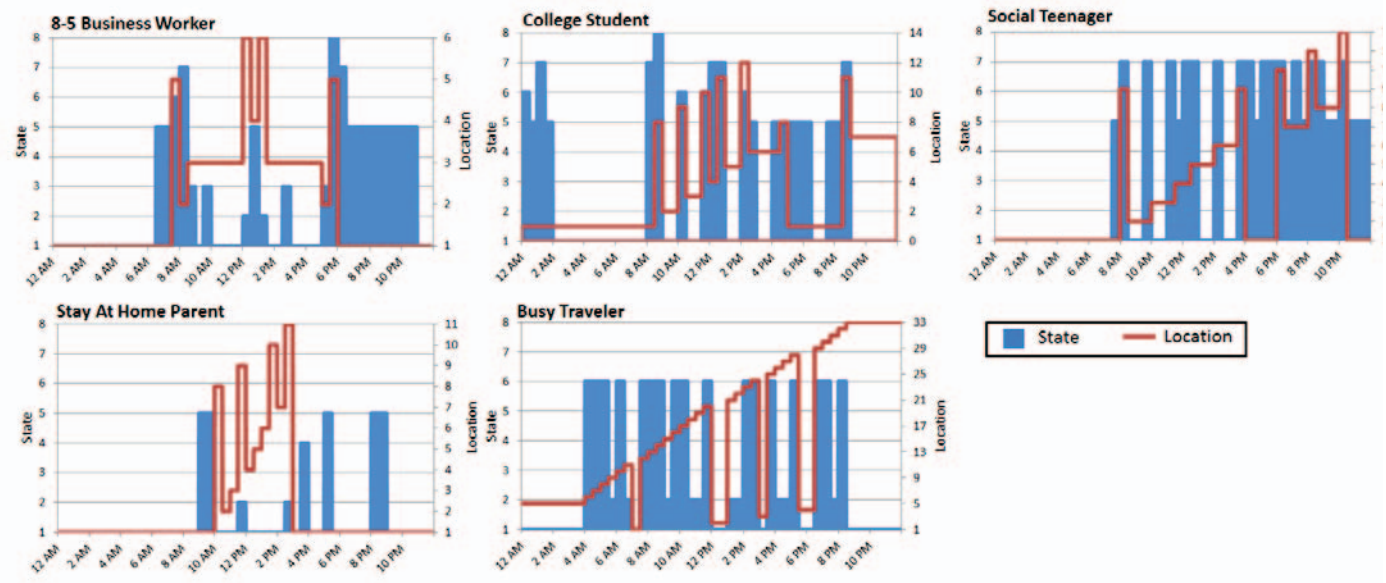


Figure 8. Synthetic user profiles

Three different neural network (NN) implementations with a varying number of hidden units, equal to the total (H=18), half (H=9), and one-sixth (H=3) the size of the attribute space were evaluated. We compared the prediction accuracy of our algorithms with the configuration prediction strategy presented in [13] (*MVSOM – Missing Values Self-Organizing Map*). As illustrated in Figure 9, Support vector machines and the application of neural networks with a number of hidden units of at least half the size of the attribute space resulted in the highest prediction rates. K-nearest neighbor (KNN), linear logistic regression (LLR), and linear discriminant analysis (LDA) also performed fairly well, with prediction accuracies in the range of 60 – 90 %. However these approaches were much more sensitive to the usage pattern. MVSOM performed the worst and had a high degree of variance in both the usage pattern and random training data selection.

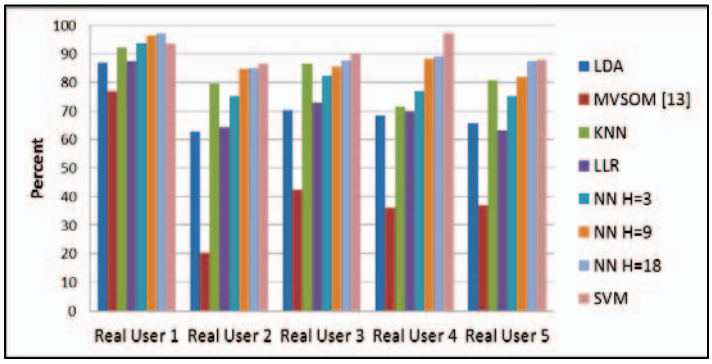


Figure 9: Real user algorithm prediction accuracy

The same algorithms were applied to the synthetic user profiles; however, the attribute space was reduced to only *Day*, *Time*, *Location*, *GPS Satellites*, *WiFi RSSI*, *Network Signal Strength*, *Data Needed*, *Coarse Location Needed*, and *Fine Location Needed*. The same strategy for selecting numbers of hidden units for the neural network implementations was applied for the reduced attribute space. Figure 10 illustrates the algorithm prediction rates for the synthetic users, which demonstrate similar trends as in the case of the real user data.

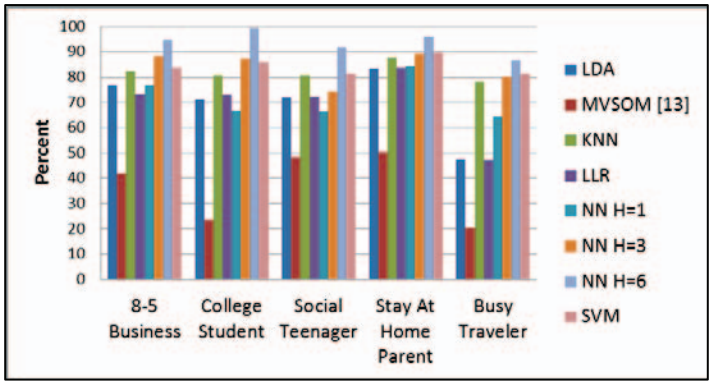


Figure 10: Synthetic user algorithm prediction accuracy

6.2 Energy Savings

It is important to note that despite high prediction accuracy, the amount of potential energy savings is still highly dependent on the user's device usage pattern and if the algorithms are positively or negatively predicting states where energy can be conserved. More complicated user patterns are more difficult for the algorithms to predict correctly. In addition, false predictions can cause either more or less energy to be consumed. For example, if an algorithm predicted state 8 when the actual state should have been state 1, negative energy savings would be achieved. Figures 11 and 12 illustrate the energy savings achieved by the individual algorithms when the algorithm's prediction target states are applied to the real and synthetic user profiles. We compare our algorithms against the VRL technique (*Variable Rate Logging* [9]). Note that as VRL does not predict system state, results for its prediction accuracy are not shown in Figure 9 presented earlier. Although simpler linear models can achieve high energy savings, it is important to note that energy savings themselves are not good discriminants of an algorithm's *goodness* because user satisfaction must also be considered. For example, if a user spends a significant amount of time in the energy consuming state 8 and the algorithms are predicting a less energy-consuming state during these instances, then more energy can be conserved at the cost of user-satisfaction. Directly correlating the prediction accuracy of the algorithms is especially important for highly interactive users such as users 2 and 5, as

opposed to minimally interactive users, e.g., user 1. All energy savings are relative to the baseline case for Android systems without proactive multi-network interface management.

Lower prediction and more generalized models result in the highest energy savings as in the case of LDA and LLR. However, again, these higher savings come at the cost of degraded user satisfaction. *SVM and KNN overall perform fairly well in terms of both prediction accuracy and energy savings potential, as does the nonlinear logistic regression with NN approach*. With the latter, an important point to note is that prediction accuracy is proportional to the complexity of the neural network and indirectly proportional to the net energy savings. This outcome is expected as less complex neural networks will result in more generalized models relaxing the constraint for inaccurate predictions that result in higher energy savings. It is also important to note that although energy savings are small for heavy users, this comes as an artifact of our optimization technique – we are exploiting windows of opportunity, which are fewer for heavy users. MVSOM, with its low prediction rates, also led to instances of negative energy savings, as it often predicted higher energy states when the true target state was one of less energy consumption. Thus we believe that the MVSOM approach is not very viable for use in mobile embedded systems. VRL's energy saving capability is constrained because it does not disable device interfaces (only deactivates location logging or reduces logging rate), ignoring idle energy consumption. *Overall, compared to VRL, the average energy savings of our KNN algorithm is 25.6%, that of our SVM algorithm is 15%, and that of our NN approaches is 11.7% (H=18), 24.1% (H=9), and 24% (H=3) for real user patterns*.

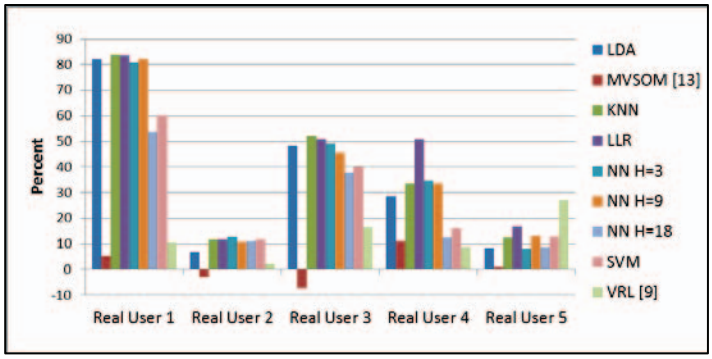


Figure 11: Percent energy saved for real users

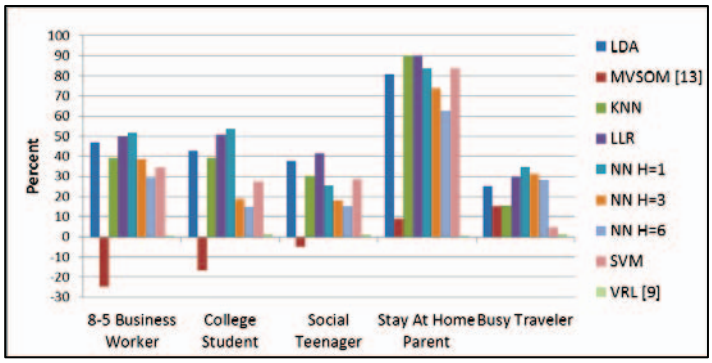
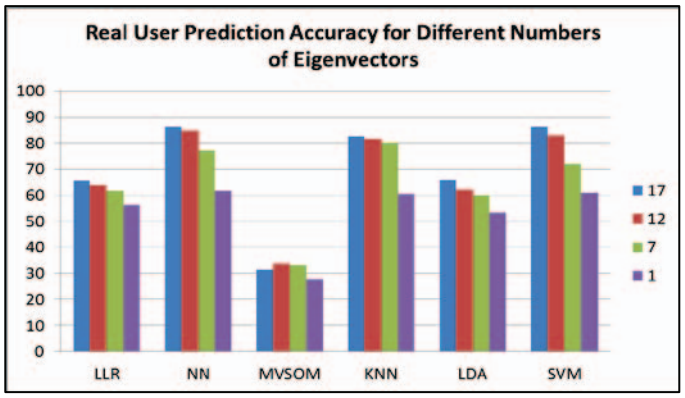
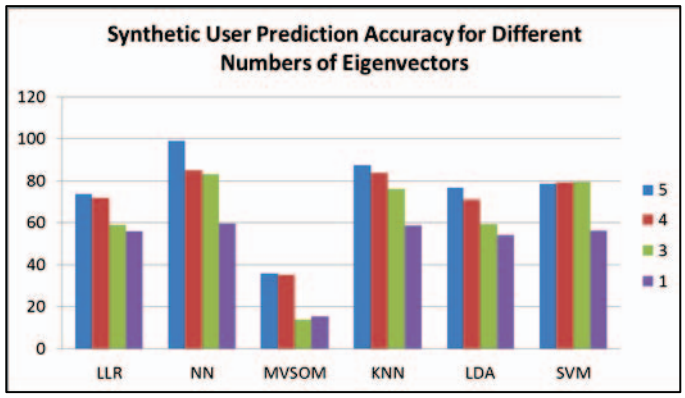


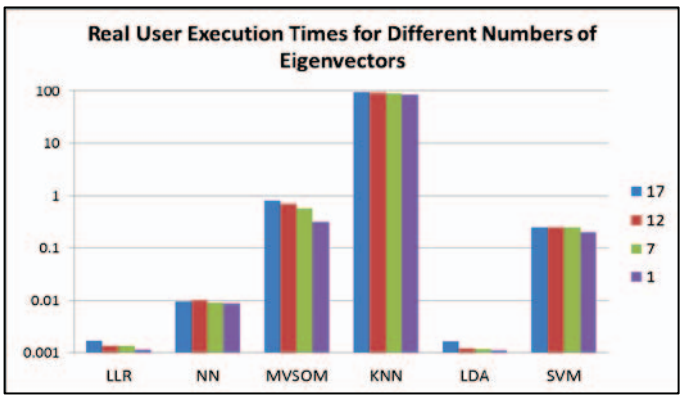
Figure 12: Percent energy saved for synthetic users



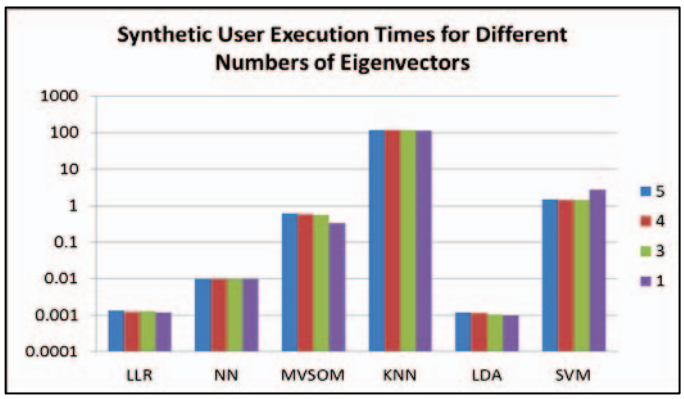
(a)



(b)



(c)



(d)

Figure 13: PCA Results

6.3 Principal Component Analysis

The complexity of the machine learning algorithms used in this work depends on the number of inputs. Because we are using a total of 19 contextual inputs, we performed Principal Component Analysis (PCA), a form of dimensionality reduction, to see how the accuracy and overhead of the algorithms was affected. PCA is known as a feature extraction method, in which a new set of *k* dimensions is created that are combinations of the original *d* dimensions [24]. PCA accomplishes this by performing an orthogonal transformation such that the directions in the data space along which the data varies the most are projected onto the most significant coordinates in a new coordinate system. In other words, the greatest variance comes to lie on the first coordinate, the second greatest variance comes to lie on the second coordinate, and so on. This transformation is motivated by the assumption that directions in the data space along which data varies least are mostly due to noise, and can be removed without loss of information. *(vi)*

The PCA results are shown in Figure 13. Figures 13 (a) and (b) show the prediction accuracy obtained by projecting the data onto different numbers of eigenvectors, effectively reducing the attribute space to the number of eigenvectors. The charts show that prediction accuracy significantly decreases as the number of eigenvectors decreases, generally. This is expected because reducing the number of input attributes simplifies the model. Figures 13 (c) and (d) show the execution (prediction) times of each algorithm after the data has been projected onto different numbers of eigenvectors. In most cases, execution times decrease slightly as the number of eigenvectors decreases. However, because of the significant degradation in prediction accuracy, the slightly reduced execution times are not enough to make PCA dimensionality reduction a viable option.

6.4 Implementation Overhead

The prediction and energy saving results presented in the previous sections were obtained using a Python implementation of the algorithms on a 2.6 GHz Intel® Core i5™ processor. When considering real-world implementation, it is important to consider the implementation overhead of the individual algorithms. Current hardware in mobile devices on the market today is quickly catching up to the abilities of modern stationary workstations (e.g. Google's Galaxy Nexus – 1.2 GHz dual-core processor). We determined the implementation overhead for our learning algorithms on the Google Nexus One with a 1 GHz Qualcomm QSD 8250 Snapdragon ARM processor [22], as shown in the third column in Table 4. The values shown in the column are average prediction times for each algorithm at runtime. The fourth and final column in the table shows actual execution times of each algorithm when run on an Nvidia Tegra 2 1.2 GHz dual-core processor. In both cases, KNN's run time is several orders of magnitude larger than any of the other algorithms, because all computations are deferred until classification. Therefore, although KNN is as good as or better than the support vector machine (SVM) and neural network (NN) based approaches in terms of energy savings and prediction, *the support vector machine approach is preferable because of its fast execution time*. The non-linear logistic regression with NN approach has a longer execution time than SVM, however, if a slightly longer execution time is acceptable, it may be preferable because of its higher energy savings potential.

Table 4: Average algorithm run times in seconds

Algorithm	Intel Core i5	Qualcomm 8520 Snapdragon	Nvidia Tegra 2
LDA	0.00139	0.00361	0.07687
LLR	0.00118	0.00307	0.06525
NN (all 3 variants)	0.00962	0.02501	0.53199
KNN	97.7428	254.131	5405.18
SVM	0.00037	0.00095	0.02021
MVSOM [13]	0.82701	2.15023	45.7337
VRL [9]	0.01977	0.05140	1.09328

In summary, our proposed LDA and LLR approaches have the lowest implementation overhead and can result in high energy savings, but often at the cost of user satisfaction. Although our KNN approach is very effective in terms of prediction accuracy and energy savings, its unreasonable implementation overhead renders it unacceptable for real-world applications. The prior work with MVSOM [13] provides low energy savings as a result of its poor prediction accuracy, and takes a long time to run; whereas VRL [9] has low run time but also very low energy savings. *Our support vector machine based approach provides good accuracy, good energy savings, and demonstrates the best adaptation to various unique user usage patterns, while maintaining a low implementation overhead.* Our non-linear logistic regression with neural network approach that uses the fast scaled conjugate gradient training method and with the number of hidden units equal to half the attribute space offers the same benefits, but with slightly higher energy savings and implementation overhead.

6.5 Real-World Implementation Considerations

When contemplating a real-world implementation, algorithm training times and data-dependent prediction accuracies must be considered. Table 5 shows the raw training and prediction times, as well as the prediction accuracies of each algorithm for various lengths of data gathering periods. We tested each algorithm using one, three, five, and seven days' worth of data. The last four columns show in the table how the prediction accuracies change with smaller amounts of data. Although accuracy certainly decreases with the amount of data, most of the algorithms perform fairly well with less than a full week of data. However, training for just one day does not provide sufficient accuracy for any of the algorithms. This is expected – a user's activity over the course of just one day is rarely indicative of their activity for the rest of the week. The first four columns of data in Table 5 show that training times also decrease with smaller amounts of data. Our KNN approach exhibited the fastest overall training time. However, this is because all computation is postponed until the prediction occurs in the KNN algorithm, which is apparent when observing its large implementation overhead in Table 4. Our LDA and LLR approaches also demonstrated fast training times, while the training times for our NN and SVM approaches were significantly slower. Although the training phase for the NN and SVM approaches takes more time than the other algorithms, we propose that such training can be performed quickly and in an energy-efficient manner without affecting user QoS whenever the device is plugged in and charging. Furthermore, these training times are only applicable to the *initial* algorithm training – training becomes less frequent over time as the algorithms continually learn the user's behavior patterns. Retraining consists of the algorithms merely making corrections to the learned models instead of learning entirely new models. The training times in Table 5 indicate how long training

Table 5: Average algorithm training/prediction times and prediction accuracies for various data gathering periods

Algorithm	Training Time (Seconds)				Prediction Time (Seconds)				Prediction Accuracy (Percent)			
	1 Day	3 Days	5 Days	7 Days	1 Day	3 Days	5 Days	7 Days	1 Day	3 Days	5 Days	7 Days
LDA	0.00187	0.00312	0.00677	0.00737	0.00032	0.00055	0.00078	0.00098	6.76	40.11	64.73	79.26
LLR (100 SCG iterations)	0.15064	0.13655	0.18087	0.23301	0.00027	0.00066	0.00071	0.00129	26.67	62.55	63.15	64.80
NN (100 SCG iterations)	1.46392	2.13049	4.36602	6.42674	0.00237	0.00272	0.00651	0.00671	31.11	84.28	87.15	88.76
KNN	0.00005	0.00044	0.00074	0.00094	0.07617	5.76673	28.2758	52.6942	71.11	79.87	83.84	86.90
SVM	0.00033	0.26553	1.33917	2.34158	0.00011	0.03022	0.14527	0.29965	11.11	86.95	88.12	90.17
MVSOM [13]	1.31357	1. 08561	1.09252	1. 11093	0.01708	0.12699	0.28049	0.37003	18.89	25.20	34.06	35.42

takes for each algorithm to reach the accuracy shown in the table. The initial training phase for each algorithm can be customized to reach a desired level of accuracy, allowing the user to control the tradeoff between speed and accuracy. For example, we chose 100 SCG iterations as the cutoff for the training in the LLR and NN algorithms. The choice of 100 iterations was determined to be a good compromise between speed and accuracy by examining the RMSE (root mean squared error) over time during the training phase. Figure 14 shows the how the RMSE (root mean squared error) for one of the real users changes for 100 and 2000 SCG iterations. It can be seen that the RMSE converges to approximately the same value when trained for 100 iterations and 2000 iterations, eliminating the need for further training after 100 iterations. For 100 SCG iterations, training took approximately 7 seconds and resulted in 88.76% accuracy, as opposed to approximately 136 seconds and 90.69% accuracy for 2000 iterations, significantly reducing training time for a very minimal loss in accuracy.

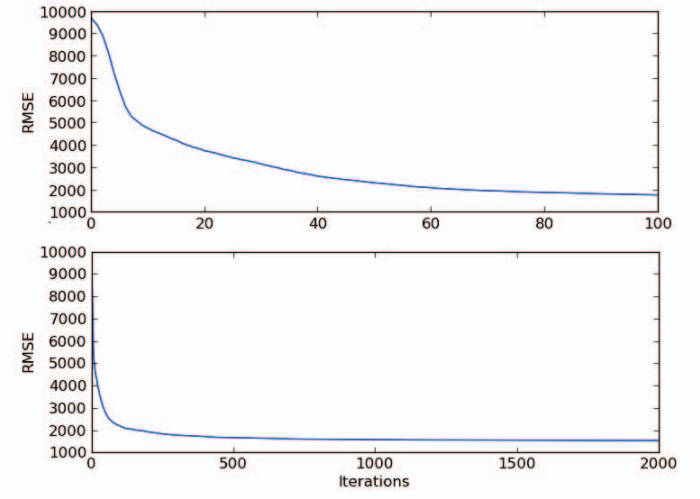


Figure 14: RMSE over 100 and 2000 SCG iterations

Another important point to notice in Table 5 is that each algorithm's prediction time increases significantly as more and more data is gathered. For this reason, data should not be stored indefinitely – data more than one week old should be deleted and replaced with more current data. This will allow the models to be quickly retrained while still maintaining acceptable prediction accuracies. Because retraining takes a trivial amount of time, it can be done on a daily basis without negatively impacting user QoS.

7 Conclusions And Future Work

In this work we demonstrated the effectiveness of using various machine learning algorithms on user spatiotemporal and device contexts in order to dynamically predict energy-efficient device interface configurations. We demonstrated up to a 90% successful prediction using support vector machines, neural networks and k-nearest neighbor algorithms, showing improvements over the self-organizing map prediction approach proposed in [13] by approximately 50%. In addition, approximately 85% energy savings was achieved for minimally active users with an average improvement of 15% energy savings compared to the variable rate logging algorithm (VRL) proposed in [9] for our best approach involving support vector machines that also has high prediction accuracy and low overhead. If slightly more implementation overhead is acceptable, our approach involving non-linear logistic regression with neural networks can provide even more energy savings, with an average improvement of 24% compared to VRL [9]. A possible extension to our work is to conduct large scale studies that recruit sample groups much larger than that considered in this work. Such large user groups could lead to the creation of user classes for which unique class-specific usage patterns could be discovered. This in turn could allow for more aggressive optimization of our framework, for instance by reducing training time and improving prediction accuracy.

References

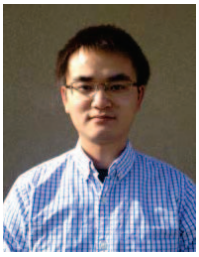
[1] H. Petander, "Energy-aware network selection using traffic estimation," in MICNET, pp. 55-60, Sept. 2009.
[2] M. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, M. J. Neely, "Energy-delay tradeoffs in smartphone applications," MobiSys, pp. 255-270, Jun. 2010.
[3] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, L. Cox, "EnLoc: energy-efficient localization for mobile phones," INFOCOM, pp. 19-25, Jun. 2009.
[4] K. Lin, A. Kansal, D. Lymberopoulos, F. Zhao, "Energy-accuracy trade-off for continuous mobile device Location," MobiSys, pp. 285-298. Jun. 2010.
[5] F. B. Abdesslem, A. Phillips, T. Henderson, "Less is more: energy-efficient mobile sensing with SenseLess," MobiHeld, pp. 61-62, Aug. 2009.
[6] J. Paek, J. Kim, R. Govindan, "Energy-efficient rate-adaptive GPS-based positioning for smartphones," MobiSys, pp. 299-314, Jun. 2010.
[7] I. Shafer, M. L. Chang, "Movement detection for power-efficient smartphone WLAN localization," MSWIM, pp. 81-90, Oct. 2010.
[8] M. Youssef, M. A. Yosef, M. El-Derini, "GAC: energy-efficient hybrid GPS-accelerometer-compass GSM localization," GLOBECOM, pp. 1-5, Dec. 2010.
[9] C. Lee, M. Lee, D. Han, "Energy efficient location logging for mobile device," SAINT, pp. 84, Oct. 2010.
[10] K. Nishihara, K. Ishizaka, J. Sakai, "Power saving in mobile devices using context-aware resource control," ICNC, pp. 220-226, 2010.
[11] Z. Zhuang, K. Kim, J. P. Singh, "Improving energy efficiency of location sensing on smartphones," MobiSys, pp. 315-330, Jun. 2010.
[12] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," MobiSys, pp. 179-192, 2009.
[13] L. Batyuk, C. Scheel, S. A. Camtepe, S. Albayrak, "Context-aware device self-configuration using self-organizing maps" OC, pp. 13-22, June 2011.
[14] T. Anagnostopoulos, C. Anagnostopoulos, S. Hadjiefthymiades, M. Kyriakakos, A. Kalousis, "Predicting the location of mobile users: a machine learning approach," ICPS, pp. 65-72, July 2009.
[15] T. Mantoro, A. Olowolayemo, S. O. Olatunji, "Mobile user location determination using extreme learning machine," ICT4M, pp. D25-D30, 2011.
[16] J. Kang, S. Seo, J. W. Hong, "Usage pattern analysis of smartphones," APNOMS, pp. 1-8, Nov. 2011

[17] B. K. Donohoo, C. Ohlsen, S. Pasricha, "AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices," ICCD, pp. 168-174, Oct. 2011.
[18] S. Choi, et al., "A selective DVS technique based on battery residual microprocessors and microsystems," Elsevier Sc., 30(1):33–42, 2006.
[19] F. Qian, et al., "TOP: tail optimization protocol for cellular radio resource allocation," ICNP, 2010.
[20] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. "Energy consumption in mobile phones: a measurement study and implications for network applications," IMC, 2009.
[21] S. Swanson, M.B. Taylor. "Greendroid: Exploring the next evolution of smartphone application processors," Communications Magazine, IEEE. Vol 49. Issue 4. April 2011.
[22] HTC, "Google Nexus One Tech Specs," http://www.htc.com/us/support/nexus-one-google/tech-specs
[23] C. M. Bishop, "Pattern Recognition and Machine Learning," 1st ed. New York: Springer Science+Business Media, 2006
[24] E. Alpaydin, "Introduction to machine learning," 2nd ed. Massachusetts: The MIT Press, 2010.
[25] M. Moller, "Efficient training of feed-forward neural networks," Ph.D. dissertation, CS Dept., Aarhus Univ., Arhus, Denmark, 1997.
[26] Android Developers, official website, http://developer.android.com/index.html.
[27] Monsoon Solutions Inc., official website, http://www.msoon.com/LabEquipment/PowerMonitor, 2008.
[28] X. Li, H. Cao, E. Chen, J. Tian, "Learning to infer the status of heavy-duty sensors for energy-efficient context-sensing," TIST, ACM, vol. 3, issue 2, no. 35, Feb. 2012.
[29] C. Chang, C. Lin, "LIBSVM – A library for support vector machines," http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[30] A. Shye, B. Scholbrock, G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," MICRO, pp. 168-178, 2009.
[31] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, J. A. Landay, "Myexperience: A system for in situ tracing and capturing of user feedback on mobile phones," MobiSys, pp. 57–70, 2007.
[32] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, D. Estrin. "Diversity in smartphone usage," MobiSys, pp. 179-194, 2010.
[33] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "LiveLab: measuring wireless networks and smartphone users in the field," HotMetrics, pp. 1-6, June 2010.
[34] C. Tossell, P. Kortum, A. Rahmati, C. Shepard, L. Zhong, "Characterizing web use on smartphones," CHI, pp. 2769-2778, May 2012.
[35] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck, "Profiling resource usage for mobile applications: a cross-layer approach," MobiSys, pp. 321-334, June 2011.
[36] J. Kang, S. Seo, J. W. Hong, "Usage pattern analysis of smartphones," APNOMS, pp. 1-8, Sept. 2011.
[37] S. Pasricha, S. Mohapatra, M. Luthra, N. Dutt, N. Subramanian, "Reducing Backlight Power Consumption for Streaming Video Applications on Mobile Handheld Devices", Embedded Systems for Real-Time Multimedia (ESTIMedia), Oct 2003.
[38] S. Pasricha, M. Luthra, S. Mohapatra, N. Dutt, N. Subramanian, "Dynamic Backlight Adaptation for Low Power Handheld Devices", IEEE Design and Test (IEEE D&T), Special Issue on Embedded Systems for Real Time Embedded Systems, Sep-Oct 2004.

Brad K. Donohoo received the B.S. degree in Computer Engineering from Utah State University in Logan, UT, in 2010, and the M.S. degree in Electrical Engineering from Colorado State University in Fort Collins, CO, in 2012. He is currently employed as a civilian Software Engineer at Hill Air Force Base in Utah. His research interests include hardware and software design of embedded systems, mobile computing, and low-power and fault-tolerant design.

Chris Ohlsen received the B.S. degree in Mechanical Engineering from University of Texas in Austin, TX, in 2008, and the M.S. degree in Electrical Engineering from Colorado State University in Fort Collins, CO, in 2012. He is currently employed in the Technology Group at Woodward in Loveland, CO. His research interests include hardware and software development of embedded systems, digital control systems, and software tool design.

Sudeep Pasricha (M'02) received the B.E. degree in electronics and communication engineering from Delhi Institute of Technology, Delhi, India, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of California, Irvine, in 2005 and 2008, respectively. He is currently an Assistant Professor of Electrical and Computer Engineering at Colorado State University, Fort Collins. His research interests are in the areas of energy efficiency and fault tolerant design for high performance computing, embedded systems, and mobile computing. Dr. Pasricha is currently an Advisory Board member of ACM SIGDA, Information Director of ACM Transactions on Design Automation of Electronic Systems (TODAES), Editor of the ACM SIGDA E-news, Organizing Committee Member and/or Technical Program Committee member of various IEEE/ACM conferences such as DAC, DATE, CODES+ISSS, NOCS, and GLSVLSI. He was the recipient of the AFOSR Young Investigator Award in 2012, and Best Paper Awards at the IEEE AICCSA 2011, IEEE ISQED 2010 and ACM/IEEE ASPDAC 2006 conferences.



Yi Xiang received the B.S. degree in Microelectronics from University of Electronic Science and Technology of China, Chengdu, China, in 2010. He is currently a Ph.D. candidate in the Electrical and Computer Engineering department at Colorado State University in Fort Collins, CO. His research interests include computer architecture, parallel embedded systems, heterogeneous computing, and CAD algorithms.

Charles Anderson received the B.S. degree in computer science from the University of Nebraska, Lincoln, in 1978 and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, in 1982 and 1986, respectively. He worked in a machine learning research lab at GTE Labs, Waltham, MA, until 1991. Since then he has been a faculty member in the Department of Computer Science at Colorado State University. He teaches core CS courses and artificial intelligence and machine learning graduate courses. His research interests are in machine learning for pattern classification, modeling and control applications.