Big Data Small Footprint: The Design of A Low-Power Classifier for Detecting Transportation Modes

Meng-Chieh Yu *
HTC, Taiwan
Mengchieh_Yu@htc.com

Tong Yu *
National Taiwan University
r01922141@csie.ntu.edu.tw

Shao-Chen Wang
HTC, Taiwan
Daniel.SC_Wang@htc.com

Chih-Jen Lin
National Taiwan University
cjlin@csie.ntu.edu.tw

Edward Y. Chang
HTC, Taiwan
Edward_Chang@htc.com

ABSTRACT

Sensors on mobile phones and wearables, and in general sensors on IoT (Internet of Things), bring forth a couple of new challenges to big data research. First, the power consumption for analyzing sensor data must be low, since most wearables and portable devices are power-strapped. Second, the velocity of analyzing big data on these devices must be high, otherwise the limited local storage may overflow.

This paper presents our hardware-software co-design of a classifier for wearables to detect a person's transportation mode (i.e., still, walking, running, biking, and on a vehicle). We particularly focus on addressing the big-data small-footprint requirement by designing a classifier that is low in both computational complexity and memory requirement. Together with a sensor-hub configuration, we are able to drastically reduce power consumption by 99%, while maintaining competitive mode-detection accuracy. The data used in the paper is made publicly available for conducting research.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology-classifier design and evaluation

General Terms

Algorithms, Design, Experimentation, Measurement

Keywords

Sensor hub, Big data small footprint, Context-aware computing, Transportation mode, Classification, Support vector machines

*These two authors contributed equally.

1. INTRODUCTION

Though cloud computing promises virtually unlimited resources for processing and analyzing big data [6], voluminous data must be transmitted to a data center before taking all the advantages of cloud computing. Unfortunately, sensors on mobile phones and wearables come up against both memory and power constraints to effectively transmit or analyze big data. In this work, via the design of a *transportation-mode detector*, which extracts and analyzes motion-sensor data on mobile and wearable devices, we illustrate the critical issue of *big data small footprint*, and propose strategies in both hardware and software to overcome such resource-consumption issue.

Detecting transportation modes (such as still, walking, running, biking, and on a vehicle) of a user is a critical subroutine of many mobile applications. The detected mode can be used to infer the user's *state* to perform context-aware computing. For instance, a fitness application uses the predicted state to estimate the amount of calories burnt. A shopping application uses the predicted state to infer if a user is shopping or dinning when she/he wanders in front of a shop or sits still at a restaurant. Determining a user's transportation mode requires first collecting movement data via sensors, and then classifying the user's state after processing and fusing various sensor signals. Though many studies (see Section 2) have proposed methods for detecting transportation modes, these methods often make unrealistic assumptions of unlimited power and resources. Several applications have been launched to do the same. However, all these applications are power hogs, and cannot be turned on all the time to perform their duties. For instance, the sensors used by *Google Now* [13] on Android phones consume around 100mA, and thus forces most users to turn the feature off. Similarly, processing sensor data on wearables[1] must minimize power consumption in order to lengthen the operation time of the hosting devices.

To minimize power consumption and memory requirement, we employ both hardware and software strategies. Though this paper's focus is on reducing the footprint of a big-data classifier, we present the entire solution stack for completeness. Our presentation first reveals bottlenecks and then accurately accounts for each strategy's contribution. Specifically on the challenges that are relevant to the big-

[1]The capacity of a battery on a typical wearable, e.g., Sony/Samsung watch, is under 315mAh [27].

data community, we employ the following four strategies to tackle them:

- *Big data.* The more data that can be collected, the more accurate a classifier can be trained.
- *Small footprint.* The computational complexity of a classifier should be low, and preferably independent of the size of training data. At the same time, model complexity must remain robust to maintain high classification accuracy. Tradeoffs between model complexity and computational complexity are carefully studied, experimented, and analyzed.
- *Data substitution.* When the data of a low power-consuming sensor can substitute that of a higher one, the higher power-consuming sensor can be turned off, thus conserving power. Specifically, we implement a virtual gyroscope solution using the signals of an accelerometer and a magnetometer, which together consumes 8% power compared with using the signals of a physical gyroscope (see Table 1 for power specifications).
- *Multi-tier design.* We design a multi-tier framework, which uses minimal resources to detect some modes, and increases resource consumption only when between-mode ambiguity is present.

By carefully considering trade-offs between model complexity and computational complexity, and by minimizing resource requirement and power consumption (via hardware-software co-design and reduction of the classifier's footprint), we reduce power consumption by 99% (from 88.5mA to 0.73mA), while maintaining 92.5% accuracy in detecting five transportation modes.

The rest of the paper is organized as follows: Section 2 presents representative related work. Section 3 depicts feature selection, classifier selection, and our error-correction scheme. In Section 4, we present the design of a small footprint classifier for a low-power sensor hub. In addition, we propose both a virtual gyroscope solution and multi-tier framework to further reduce power consumption. Section 5 outlines our data collection process and reports various experimental results. The transportation-mode data is made available at [15] for download. We summarize our contributions and offer concluding remarks in Section 6.

2. RELATED WORK

Prior studies on transportation-mode detection can be categorized into three approaches: *location-based*, *motion-sensor-based*, and *hybrid*. The key difference of our work is that we address the practical issue of resource consumption.

2.1 Location-Based Approach

The location-based approach is the most popular one for detecting transportation modes. This is because sensors such as GPS, GSM, and WiFi are widely available on mobile phones. In addition, the location and changing speed can conveniently reveal a user's means of transportation.

The method of using the patterns of signal-strength fluctuations and serving-cell changes to identify transportation modes is proposed by [2]. The work achieves 82% accuracy in detecting among modes of still, walk, and on a vehicle. For the usage of GSM data, the study of [26] extracts mobility properties from a coarse-grained GSM signal to achieve 85% accuracy for detecting among the same three modes. The work of [35] extracts heading change rate, velocity, and acceleration from GPS signals to predict the modes of walk, bike, driving, and bus. Recent work of [30] uses GPS and knowledge of the underlying transportation network including real time bus locations, spatial rail and spatial bus stop information to achieve detection accuracy of 93%.

Table 1: Power consumption of processors and sensors. The active status indicates that only our transportation-mode algorithm is running.

	Power	Condition
CPU (running at 1.4GHz)	88.0mA	Active status
	5.2mA	Idle status
MCU (running at 16MHz)	0.5mA	Active status
	0.1mA	Idle status
GPS	30.0mA	Tracking satellite
WiFi	10.5mA	Scanning every 10 sec
Gyroscope	6.0mA	Sampling at 30Hz
Magnetometer	0.4mA	Sampling at 30Hz
Accelerometer	0.1mA	Sampling at 30Hz

Unfortunately, the location-based approach suffers from high power consumption and can fail in environments where some signals are not available (e.g., GPS signals are not available indoors). Table 1 lists power consumption of processors and sensors. It is evident that both GPS and WiFi consume significant power, and when they are employed, the power consumption is not suitable for devices such as watches and wrist bands, whose 315mA batteries last less than half a day when only the GPS is on.

2.2 Sensor-Based Approach

The motion-sensor-based approach is mostly used to detect between walking and running in commercial products such as Fuelband [22], miCoach [1], and Fitbit [9]. The study of [31] uses an accelerometer to detect six typical transportation modes, and concludes that the acceleration synthesization based method outperforms the acceleration decomposition based method. The research of [34] extracts orientation-independent features from vertical and horizonal components and magnitudes from the signals of an accelerometer. Combined with error correction methods using k-means clustering and HMM-based Viterbi algorithm, this work achieves 90% accuracy for classifying six modes.

2.3 Hybrid Location/Sensor-Based Approach

For the location-based and motion-sensor-based hybrid approach, the studies of [32], [25], and [17] employ both GPS and accelerometer signals to detect the transportation mode. In addition, [18] proposes an adaptive sensing pipeline, which switches the depth and complexity of signal processing according to the quality of the input signals from GPS, an accelerometer, and a microphone. However, these schemes all suffer from high power consumption.

2.4 Resource Consideration

The tasks of sensor-signal sampling, feature extraction, and mode classification are continuously run to consume resources. Some prior works address the problem of power consumption via signal subsampling and process admission control. The studies of [24] focus on adapting the sampling rate to extract sensor signals. The data admission control and duty cycling strategies are proposed by [18]. The work of [28] presents a framework that reduces the need of running

Table 2: Representative work of transportation-mode detection (accuracy in percentage). Note that Acc means accelerometer and mic means microphone in this table.

Ref	# Modes	Sensors Used	Accuracy	Power
[2]	3	GSM	82.00%	*not considered*
[26]	3	GSM	84.73%	*not considered*
[35]	4	GPS	76.20%	*not considered*
[30]	6	GPS, GIS	93.50%	*not considered*
[20]	3	GSM, WiFi	88.95%	*not considered*
[31]	6	Acc.	70.73%	*not considered*
[34]	6	Acc.	90.60%	*not considered*
[32]	4	Acc., GPS	91.00%	*not considered*
[18]	5	Acc., GPS, mic.	95.10%	20.5mA
[25]	5	Acc., GPS, GSM	93.60%	15.1mA

the main recognition system and can still maintain competitive accuracy. The work of [21] reduces power consumption by inferring unknown context features from the relationship between various contexts.

Our work provides resource management in both hardware and software, and achieves much more significant resource conservation compared with all prior approaches. Our design uses MCU to replace CPU, and low-power sensors such as accelerometer and magnetometer to replace GPS, WiFi, and gyroscope. Furthermore, the small footprint of our classifier reduces both memory requirement and power consumption.

Table 2 summarizes representative schemes mentioned in this section, including signal sources, number of detection modes, accuracy, and power consideration. (Note that accuracy values in these studies are obtained upon different datasets and experimental environments.) Most schemes do not address the power consumption issue. The ones that consider the power issue consume at least 20 times (15.1mA achieved by [25] vs. 0.73mA by ours) our proposed hardware-software co-design.

3. ARCHITECTURE

This section presents our transportation-mode detection architecture, of which the design goal is to achieve high detection accuracy at low power consumption. The architecture consists of computation modules located at *sensor hub*, *mobile client*, and *cloud server*. The sensor hub employs low-power components and makes a preliminary prediction on a user's transportation mode. The mobile client and cloud server then use additional information (e.g., location information, map, and transit route), if applicable, to further improve prediction accuracy. The three tiers work in tandem to adapt to available resources and power. In this paper, we particularly focus on depicting our design and implementation of a low-power, low-cost *sensor hub*.

The overall structure of the transportation-mode detection system on the sensor hub is depicted in Figure 1, which shows source sensors on the left-hand side and five target modes on the right-hand side. The hub employs an MCU (operating at speeds up to 72MHz), which has power consumption of 0.1mA while running at 16MHz, compared to 88.5mA of a 1.4GHz Quad-core CPU. The hub is configured with three motion sensors: an accelerometer, a gyroscope, and a magnetometer, all running at 30Hz sampling rate.

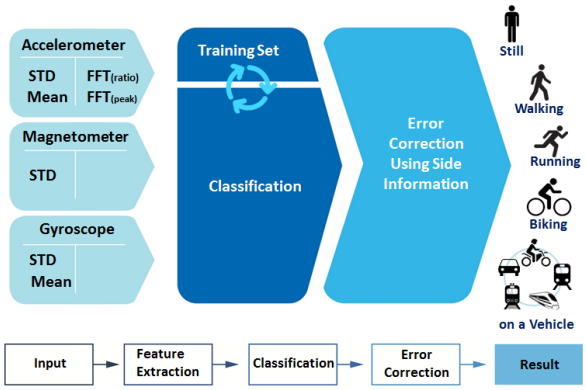


Figure 1: Structure of overall system.

Predicting transportation mode consists of three key steps: feature extraction, mode classification, and error correction. The mode classifier is trained offline via a training pipeline, details of which are presented in Section 3.2. Once signals have been collected from the sensors, the hub first extracts essential features. It then inputs the features to the classifier to determine the transportation mode. In the end, the hub performs an error correction scheme to remove noise. In this work, our target transportation modes are: still, walking, running, biking, and (on) vehicle. We use symbols Still, Walk, Run, Bike, and Vehicle to denote these modes, respectively. Note that the Vehicle mode includes motorcycle, car, bus, metro, train, and high speed rail (HSR).

The remainder of this section describes these three online steps.

3.1 Feature Extraction

Our sensor hub uses 3-axis motion sensors. An accelerometer is an electromechanical device measuring acceleration forces in three axes. By sensing the amount of dynamic acceleration, a subroutine can analyze the way the device is moving in three dimensions. A gyroscope measures angular velocity in three axes. The output of a gyroscope tells device rotational velocity in three orthogonal axes. Since a sensor hub may be mounted on a mobile device in a tilted angle, and that device can be carried by the user in any orientation, it is not productive to consider acceleration values in three separate axes. Instead, for the purpose of discerning transportation modes, the magnitude of acceleration, or the energy of motion, is essential. Therefore we combine signals from the three axes as the basis of the magnitude feature. For example, the magnitude of the accelerometer is $A_{\text{mag}} = \sqrt{(A_x)^2 + (A_y)^2 + (A_z)^2}$. The same calculation is used for signals from the gyroscope and magnetometer. This formulation enables our system to assume a random orientation and position of a device for mode prediction.

In addition, the magnitude measured at a time instant is not a robust feature. We thus aggregate signals in a moving window, and extract features from each window. We evaluated the performance by using different window sizes, and finally determined 512 as the choice because it yielded the most suitable result. (Section 5.2.1 presents the detailed evaluation.) Extracted features can be classified into two categories, including a category in the time domain and the other in the frequency domain. According to experience of prior works, we tried 22 features in the time-domain, and 8

features in the frequency domain to be the baseline for evaluation. The features that we evaluated in the time-domain include mean, magnitude, standard deviation, mean crossing rate, and covariance. The features that we evaluated in the frequency-domain include entropy, kurtosis, skewness, the highest magnitude frequency, the magnitude of the highest magnitude frequency, and the ratio between the largest and the second largest FFT (Fast Fourier Transform) values. These features in the frequency domain were calculated over frequency domain coefficients on each window of 512 samples. Finally, five time-domain and two frequency-domain features were selected after rigorous experiments. Five features in the time domain include the mean and standard deviation calculated from the signals of an accelerometer and gyroscope, and the standard deviation from a magnetometer. Two frequency-domain features include the highest magnitude frequency and the ratio of the highest and second magnitude frequency from FFT spectrum. These features are summarized as follows:

1. acc_std: standard deviation of the magnitude of accelerometer.
2. acc_mean: mean of the magnitude of accelerometer.
3. acc_FFT (peak): the index of the highest FFT value, which indicates the dominated frequency of the corresponding mode.
4. acc_FFT (ratio): the ratio between the largest and the second largest FFT values, which roughly depicts if the FFT value distribution is flat.
5. mag_std: standard deviation of the magnitude of magnetometer.
6. gyro_std: standard deviation of the magnitude of gyroscope.
7. gyro_mean: mean of the magnitude of gyroscope.

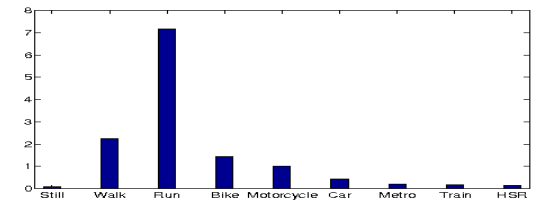
Figure 2 shows the average value of each feature on nine transportation modes. We can see that the average values of different modes vary significantly. Therefore, our selected features are effective for telling apart different transportation modes.

3.2 Classifier Selection

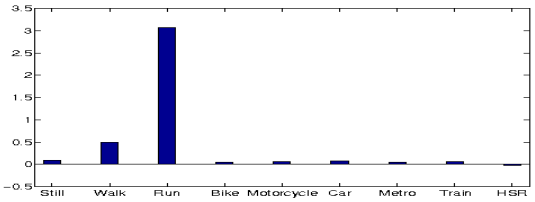
We consider three classifiers according to *model complexity*. A model of low complexity such as a linear model may underfit the data, whereas a high complexity model may suffer from overfitting. In general, we can choose an array of models, and use cross-validation to select the one that achieves the best classification accuracy on unseen data. However, we must also consider power consumption and memory use, and therefore, computational complexity and model-file size in our selection process.

We first selected three widely-used baseline models that cover the spectrum of model complexity; they are *decision tree*, *AdaBoost*, and *SVMs*. Decision tree is a piece-wise linear model, AdaBoost smooths the decision boundary of decision trees, and SVMs can work with a kernel to adjust its model complexity. For decision tree, we used the J48 implementation in the package Weka [14], which is based on the C4.5 algorithm [23]. However, a full tree can severely overfit the training data, so by default a post-pruning procedure is conducted to remove some nodes. J48 prunes nodes until the estimated accuracy is not increased.

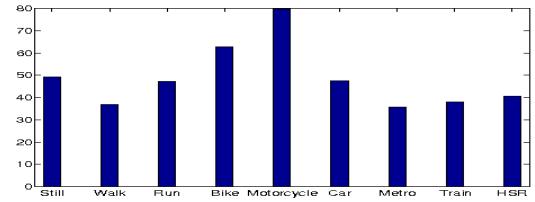
AdaBoost (Adaptive Boosting) [10] sequentially applies a classifier (called weak learner) on a weighted data set obtained from the previous iteration. Higher weights are im-



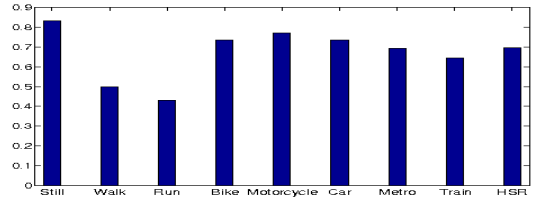
(a) Standard deviation of accelerometer values (acc_std)



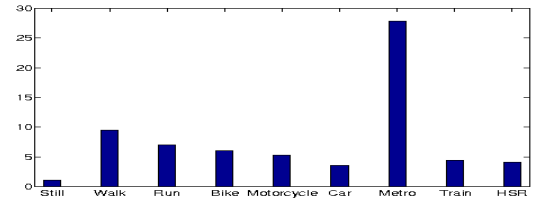
(b) Mean of accelerometer values (acc_mean)



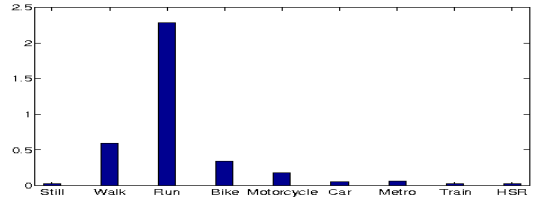
(c) Index of the highest FFT value (acc_FFT (peak))



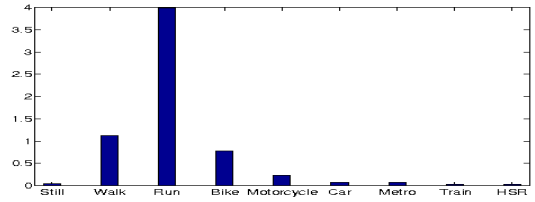
(d) Ratio of FFT values (acc_FFT (ratio))



(e) Standard deviation of magnetometer values (mag_std)



(f) Standard deviation of gyroscope values (gyro_std)



(g) Mean of gyroscope values (gyro_mean)

Figure 2: Average feature values in different modes.

posed on wrongly predicted data in previous iterations. This adaptive setting is known to mitigate overfitting. We considered J48 as the weak learner and applied the AdaBoost implementation in Weka. The number of iterations was chosen to be 10, so 10 decision trees are included in the model.

Support vector machines (SVMs) [3, 8] can efficiently perform a non-linear classification using the kernel trick, implicitly mapping data in the input space onto some high-dimensional feature spaces. It is known that SVMs are sensitive to the numeric range of feature values. Furthermore, some features in our data have values in a very small range. Therefore, we applied a log-scaling technique to make values more evenly distributed. In addition, SVMs are also known to be sensitive to parameter settings, so we conducted cross validation (CV) to select parameters. The SVM package LIBSVM [5] was employed for our experiments.

SVMs were chosen to be our classifier because SVMs enjoy the highest mode-prediction accuracy. Section 5.2.3 presents the details of our classifier evaluation. When training data is abundant (in a big-data scenario), the large number of support vectors can make the footprint of its model file quite large. In Section 4, we will address the issues of resource conservation for SVMs to drastically reduce its footprint.

3.3 Error Correction via Voting

In transportation-mode detection, an incorrect detection by a classifier may be caused by short-term changes of the transportation mode. When an activity is changed from one mode (e.g., Walk) to another (e.g., Still), the moving window that straddles the two modes during transition can include features from both modes. Therefore, the classification may be erroneous. We propose a voting scheme to address this problem.

At time $t-1$, the system maintains scores of all modes.

$$\text{score}_{t-1}(\text{Still}),\ \text{score}_{t-1}(\text{Walk}), \ldots,\ \text{score}_{t-1}(\text{Vehicle}). \quad (1)$$

Then at time t, the classifier predicts a label. This label and the past scores in (1) are used together to update scores and make mode predictions.

Figure 3 presents the pseudo code of the voting scheme. In the beginning, all of the scores are set as zero. At each time point, the score of the mode predicted by the classifier is increased by one unit. In contrast, the scores of the other modes are decreased by one unit. The upper-bound of the score is set as four units, whereas the lower bound zero. Then, the mode with the highest score is determined as the modified prediction. Section 5.2.4 presents the improvement in prediction accuracy achieved by this voting scheme.

Algorithm 3.1: VOTINGSCHEME(*cresult*)

Input
cresult : *the result detected by classifier*
Output
vresult : *the result after voting scheme*

```
max = cresult
if prob[cresult] ≤ 4
  then prob[cresult] ← prob[cresult] + 1
for i ← 0 to classNum
    do { if i ≠ cresult and prob[i] ≤ 0
           then prob[i] ← prob[i] − 1
       { if prob[max] ≤ prob[i]
           then max ← i
vresult ← max
```

Figure 3: Pseudo code of voting scheme.

4. RESOURCE CONSERVATION

As presented in Section 1, we employ three strategies: i) small footprint, ii) data substitution, and iii) multi-tier design, to conserve resources even though we use a large-pool of training instances. We present details in this section.

4.1 Small Footprint

Moving CPU computation to a low-power MCU saves significant power. However, that saving is not good enough for wearables, and we must save even more. At the same time, the low-cost sensor hub demands us to design a small-footprint classifier that does not take up much memory space. A small footprint design of our classifier not only saves memory space, but also reduces computation and thereby saves even more power.

This section aims to reduce the model-file size of SVMs. Typically, employing the kernel trick provides higher model complexity to the SVM classifier to yield higher classification accuracy. However, the kernel trick requires the formulation of a kernel matrix, the size of which depends on the size of the training data. In the end of the training stage, the yielded support vectors are collected in a model file to perform class prediction on unseen instances. The size or footprint of the model file typically depends on the size of the training data. In a big-data setting when a huge amount of training data is used to improve classification accuracy, the model file is inevitably large. Such consequence is not an issue when classification is performed in the cloud, where both memory and CPU are virtually unlimited. On our sensor hub, which is mounted on a low-power device, a large footprint is detrimental.

Let us visit the formulation of SVMs in a binary classification setting. Support vector machines [3, 8] minimize the following weighted sum of the regularization term and training losses for the given two-class training data $(y_1, \boldsymbol{x}_1), \ldots (y_l, \boldsymbol{x}_l)$.

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \max(1 - y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b), 0), \quad (2)$$

where $y_i = \pm 1, \forall i$ are class labels, $\boldsymbol{x}_i \in R^n, \forall i$ are training feature vectors, and C is the penalty parameter. An important characteristic of SVMs is that a feature vector $\boldsymbol{x}_i$ can be mapped to a higher dimensional space via a projection function $\phi(\cdot)$ to improve class separability. To handle the high dimensionality of the vector variable $\boldsymbol{w}$, we can apply the kernel method and solve the following dual optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \\ & 0 \le \alpha_i \le C, i = 1, \ldots, l, \end{aligned} \quad (3)$$

where

$$Q_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad (4)$$

and $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the kernel function. The optimal solutions of the primal problem (2) and the dual problem (3) satisfy the following relationship (l denotes the number of training instances):

$$\boldsymbol{w} = \sum\nolimits_{i=1}^{l} \alpha_i y_i \phi(\boldsymbol{x}_i). \tag{5}$$

The set of support vectors is defined to include $\boldsymbol{x}_i$ with $\alpha_i > 0$, since $\alpha_i = 0$ implies that $\boldsymbol{x}_i$ is inactive in (5). With a special $\phi(\cdot)$ function, the kernel value $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be easily calculated even though it is the inner product of two high dimensional vectors. Let us consider the following RBF (Gaussian) kernel as an example.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2},$$

where γ is the kernel parameter. RBF kernel in fact maps original data to infinite dimensions, so the SVM model (5) also has infinite dimensions and cannot be saved directly. In order to save the model, all the support vectors $\boldsymbol{x}_i$ and their corresponding non-zero α_i must be stored and then loaded into memory when making predictions. If the number of support vectors is $\mathcal{O}(l)$, then the $\mathcal{O}(ln)$ size of the model, where n is the number of features, can be exceedingly large.

In order to achieve extremely small model size, we propose an advanced setting by low-degree polynomial mappings of data [7] when storing the model. From (4), every valid kernel value is the inner product between two vectors. We consider the polynomial kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\gamma \boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^d = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j),$$

where d is the degree. If n is the number of features, then

$$\text{dimensionality of } \phi(\boldsymbol{x}) = \binom{n+d}{n}.$$

For example, if $d = 2$, then

$$\begin{aligned}\phi(\boldsymbol{x}) = [1, \sqrt{2\gamma}x_1, \ldots, \sqrt{2\gamma}x_n, \gamma x_1^2, \ldots,\\ \gamma x_n^2, \sqrt{2}\gamma x_1 x_2, \ldots, \sqrt{2}\gamma x_{n-1} x_n]^T.\end{aligned}$$

Therefore, if the dimensionality of $\phi(\boldsymbol{x})$ is not too high, then instead of storing the dual optimal solution and the support vectors (i.e., all $\alpha_i, \boldsymbol{x}_i$ with $\alpha_i > 0$), we only store $(\boldsymbol{w}, b)$ in the model. To be more precise, if $\phi(\boldsymbol{x})$ is very high dimensional (possibly infinite dimensional), then $\boldsymbol{w}$ in (5) cannot be explicitly expressed and we must rely on kernel techniques. In contrast, if $\phi(\boldsymbol{x})$ is low dimensional, then the vector $\boldsymbol{w}$ can be explicitly formed and stored. Because the length of $\boldsymbol{w}$ is the same as that of $\phi(\cdot)$, a nice property is that the model size becomes *independent* of the number of training data. If we use the one-against-one strategy for k-class data (discussed later in this section) and assume single-precision storage, the model size is

$$\begin{aligned}&\binom{k}{2} \times (\text{length of } \boldsymbol{w} + 1) \times 4\text{bytes}\\ =&\binom{k}{2} \times \left(\binom{n+d}{d} + 1\right) \times 4\text{bytes}.\end{aligned} \tag{6}$$

We hope that when using a small d (i.e., low-degree polynomial mappings), $\binom{n+d}{d}$ is smaller than $\mathcal{O}(ln)$ of using kernels and the model size in (6) is smaller. Then we can simultaneously reduce the model size and achieve nonlinear separability. Take $d = 3$ as an example.

$$\binom{n+d}{d} \le n^3 \ll ln \text{ if } n^2 \ll l. \tag{7}$$

Because the number of features (7 in our case) is small and in a big-data scenario, l can be in the order of millions, easily $n^2 \ll l$ and saving in model size is significant.

Of course, to reduce the model size as small as possible, we can simply set $\phi(\boldsymbol{x}) = \boldsymbol{x}$ so that the data is not mapped to a different space. The model size is very small because storing on $(\boldsymbol{w}, b)$ needs only $n + 1$ float-point values, where n is the number of features in linear space. However, linear SVMs' result usually cannot match that of SVMs with kernel.

Usually, when the degree of polynomial mapping increases, the performance of the SVM classifier would become better. At the same time, the SVM model size would increase. What we need to do is to find an appropriate d, which achieves a good balance between classification accuracy and model size. In Section 5.2.3, we report that degree-3 polynomial mappings achieves the best such balance.

We briefly discuss multi-class strategies because the number of transportation modes is more than two. Both decision tree and AdaBoost can directly handle multi-class data, but SVMs do not. Recall that in problem (2), $y_i = \pm 1$ so only data in two classes are handled. Here we follow LIBSVM to use the one-against-one multi-class strategy [16]. For k classes of data, this method builds a model for every two classes of training data. In the end $k(k-1)/2$ models are generated. Another frequently used technique for multi-class SVMs is the one-against-rest strategy [4], which constructs only k models. Each of the k models is trained by treating one class as positive and all the rest as negative. Although the one-against-rest method has the advantage of having fewer models,[2] its performance may not be always as good as or better than one-against-one [12]. From our experiments on mode detection, one-against-rest gives slightly lower accuracy, so we choose the one-against-one method for all subsequent SVM experiments.

For example, considering a five-class one-against-one SVM classification problem, if the number of feature is 7 and the degree of polynomial mapping 3, the dimensionality of $\phi(\cdot)$ is $\binom{n+d}{d} = \binom{7+3}{3} = 120$. After considering the bias term, the number of dimensions is 121. With $5 \times (5-1)/2 = 10$ binary SVM models, the model size would be $10 \times 121 \times 4 = 4,840$bytes $= 4.84$KB, which is smaller than the sensor hub's limit of 16KB, satisfying the memory constraint.

Another merit of our polynomial SVM model is that it requires low computational complexity when making predictions, compared with what RBF kernel SVMs do. It is especially important when the device's computing capacity is not that high. Instead of computing the decision value by

$$f(\boldsymbol{x}) = \sum\nolimits_{i=1, i\neq 0}^{l} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{8}$$

our method calculates the decision value simply by $f(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$. Therefore, for data with number of instances

[2] If using the one-against-rest multi-class strategies, only k rather than $\binom{k}{2}$ vectors of $(\boldsymbol{w}, b)$ must be stored.

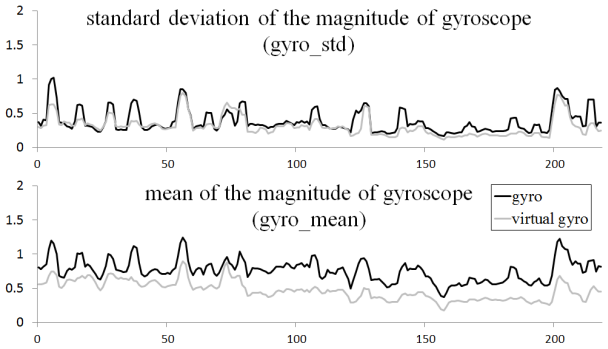


Figure 4: Comparison of the mean and standard deviation in each window from gyro and virtual gyro.

l and dimensions n, compared to the computational complexity $\mathcal{O}(ln)$ depicted in (8),[3] the computational time as well power is significantly reduced following a similar reason explained in (7).

4.2 Virtual Gyroscope Solution

A gyroscope is a device for measuring orientation, based on the principles of angular momentum. In our feature selection experiment shown in Table 5, we can see that the features of gyro_std and gyro_mean perform well especially for the mode of Bike.

However, the gyroscope takes up about 85% of the total power consumption (i.e., about 7.0mA per hour under the sensor hub environment, reported in Section 5.3.1). To further reduce power consumption, we apply a method called virtual gyroscope to simulate the data of the gyroscope from that of the accelerometer and magnetometer combined.

In the procedure of a virtual gyroscope, we first pass accelerometer data into a low-pass filter to extract the gravity force. Then, a mean filter is applied to both gravity-force and magnetometer data to reduce noise. After that, the rotation matrix, which transforms gravity-force and magnetic data from the device's coordinate system to the world's coordinate system, is computed. Finally, with the idea in [29], the angular velocity ω can be obtained by the following equation:

$$\omega = \frac{\mathrm{d}R(t)}{\mathrm{d}t}R^T(t) = \frac{1}{\Delta t}(I - R(t-1)R^T(t)),$$

where $R(t)$ is the rotation matrix at time t and Δt is the time difference. The virtual gyroscope data can be calculated using two consecutive rotation matrices and their recorded-time difference.

In this study, the features generated by the gyroscope include gyro_std and gyro_mean. Figure 4 shows the comparison of the mean and standard deviation features extracted from the physical and virtual gyroscope. The data in the figure show that though the physical and virtual gyroscope do not produce identical values, their spiking patterns are in tandem. The virtual gyroscope can achieve the same mode-prediction accuracy as the physical gyroscope at reduced power consumption. The overall evaluation of power consumption and test accuracy is shown in Section 5.3.2.

4.3 A Two-Tier Framework

The goal of our detector is to identify the modes of Still, Walk, Run, Bike, and Vehicle. However, a device is often

[3]Assume the number of support vectors is $\mathcal{O}(n)$.

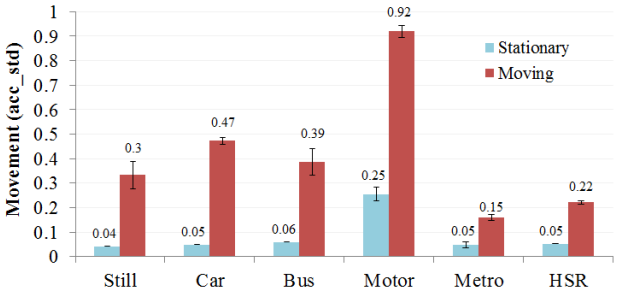


Figure 5: The comparisons of acc_std in still and five vehicle modes. Two conditions are considered for Still: the phone is on the body or not. Two conditions are considered for five Vehicle modes: the Vehicle is stationary or moving.

placed on a stationary surface. For instance, a phone is placed on a desk at home or at work in order to charge its battery. Such *fully still* mode indicates that a device is at a stationary place affected by nothing but the gravity. (Note that the *fully still* mode is a special case of the mode of Still.) If a simple rule can be derived to determine the *fully still* mode, the system will only need to extract a small number of relevant features to efficiently confirm its being in fully still. In other words, it is not necessary to activate the full mode-classification procedure at all times.

We would like to find a value of acc_std, beneath which we can safely classify the mode to be *fully still* without extracting features and running the classifier. Since from Figure 2 we can observe that acc_std is significantly smaller in mode Still than in modes Walk, Run, and Bike, we only had to examine acc_std between modes Still and Vehicle.

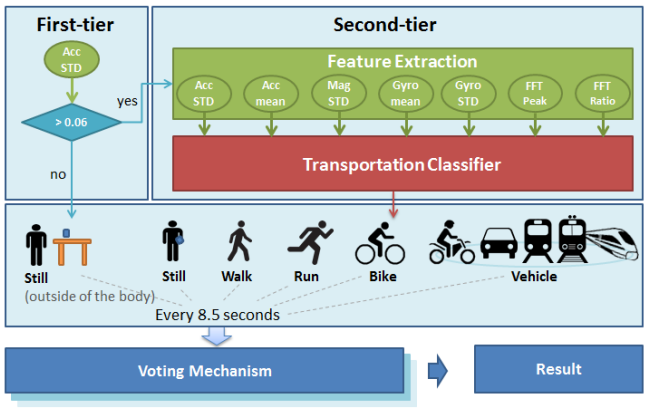


Figure 6: A hierarchical setting of transportation-mode detection.

Next, we observe the acc_std value in the different vehicle modes including motorcycle, car, bus, metro, and high speed rail (HSR). In addition, we monitor two conditions of these vehicles, including stationary Vehicle and moving Vehicle. Figure 5 compares the acc_std values in Still and five Vehicle modes. The results show that value of acc_std is lower than 0.06 except for in the motorcycle mode when the Vehicle is stationary (e.g., stopping at a traffic light). Thus, we can set 0.06 as the threshold of the two-tier framework. An ultra-low-power microchip inside the accelerometer constantly collects data and directly predicts the Still mode when the movement of the phone is insignificant. When

the accelerometer detects movement, it activates the processor on the sensor hub to run the full feature-extraction and class-prediction pipeline. Figure 6 depicts the framework, where the upper-left component activates feature extraction and mode classification according to the following rule:

If acc_std ≤ 0.06
    predict mode Still
Else
    predict a mode by the transportation-mode classifier

5. EMPIRICAL STUDY

During the design of our transportation-mode classifier, we evaluated several feature-set and model alternatives. This section documents all evaluation details, divided into three subsections: experiment setup, parameter and classifier selection, and resource conservation.

5.1 Experiment Setup

This subsection describes the details of experiment environment, and training data collection.

5.1.1 Hardware & Software

As a testing platform, we used the HTC One mobile phone, which runs Android system. HTC One is equipped with a sensor hub, which consists of an ARM Cortex-M4 32-bit MCU (operating at speeds up to 72MHz), 32KB RAM, 128KB flash, and three motion sensors: an accelerometer, a magnetometer, and a gyroscope. The processor of the sensor hub also supports digital signal processing (DSP) to enhance the processing performance for complex mathematical computations, such as FFT. It also supports fixed-point processing to optimize the usage of memory and computing time. To evaluate different parameter settings of candidate classifiers, the Weka Machine Learning tool [14] and LIBSVM library [5] were employed. To evaluate the power consumption, we used Monsoon power monitor [19].

5.1.2 Data Collection

Table 3 shows the amount and distribution of the movement data that we have collected since 2012. The 8,311 hours of 100GB data were collected via two avenues: our university program with 150 participating students, and a group of 74 employees and interns. We made sure that the pool of participants sufficiently covered different genders (60% male), builds, and ages (20 to 63 years old). We implemented a data collection Android application for participants to register their transportation status into ten modes: Still, Walk, Run, Bike, (riding) Motorcycle, Car, Bus, Metro, Train, and high speed rail (HSR). In our system design, we combine modes of motorcycle, car, bus, metro, train, and HSR into the Vehicle mode. Experiments were conducted through splitting the data from the internal program into training and testing sets.

Because the class of Vehicle includes several transportation modes and contains much more data instances than the other modes, we randomly sampled 25% of the Vehicle data to conduct training for avoiding the potential prediction bias caused by imbalanced data [33]. Because of the randomness nature in the sampling process, we prepared ten (training, test) pairs for experiments and averaged the results of our ten runs on all experiments.

Table 3: Data collection time (hours) and mode distribution.

	Internal Program	University Program
Still	107	1,750
Walk	121	1,263
Run	61	88
Bike	78	61
Motorcycle	134	1,683
Car	209	558
Bus	69	1,248
Metro	95	289
Train	67	267
HSR	91	72
Total	1,031	7,280

Table 4: Window size selection. The test accuracy, memory usage, and response time are compared.

Performance \ Window size	256	512	1,024	2,048
Accuracy	89.51	90.66	91.55	91.69
Memory usage (KB)	1	2	4	8
Response time (sec)	8.5	17.1	34.1	68.3
Latency (50% overlap)	4.3	8.6	17.1	34.2

5.2 Parameter and Classifier Selection

Experiments were conducted to set feature-extraction window size, select most effective features, and evaluate three candidate classifiers under various parameter settings.

5.2.1 Window Size

Different window sizes affect classification accuracy, response time (latency), and memory size. Too small a window may admit noise, and too large a window may overly smooth out the data. We relied on cross-validation to select a window size that achieves a reasonable tradeoff between the three factors. Besides accuracy, response time affects user experience, as the larger the window size, the longer the latency for a user to perceive a mode change. Since the sampling rate of sensors is at 30Hz, a window size of 2,048 takes over a minute to collect and then generate FFT features. Though we employ a 50% window overlapping scheme, an over half-a-minute latency is not an acceptable user experience. We consider an acceptable latency to be under ten seconds.

Table 4 reports results of applying SVMs with a degree-3 polynomial mapping on four different window sizes (256, 512, 1,024, and 2,048). The window sizes were selected as a power of two because of the FFT processing. Both sizes of 512 and 1,024 provide a reasonable response time and mode-prediction accuracy. Further increasing window size does not yield significant accuracy improvement (as expected), at the expense of long latency. We set the window size to be 512 because of its lower latency of 8.6 seconds and relatively high mode-prediction accuracy.

5.2.2 Feature Selection

Five features in the time domain (F_{time}) and two features in the frequency domain (F_{freq}) were presented and discussed in Section 3.1. Here, we report our justification of selecting those features.

We use two criteria for selecting features: effectiveness and cost. For effectiveness, we want to ensure that a feature

Table 5: Feature selection by using different feature combinations. F_{time} includes acc_std, acc_mean, mag_std, gyro_std, and gyro_mean. F'_{time} includes acc_std, acc_mean, mag_std. F_{freq} includes acc_FFT (peak) and acc_FFT (ratio).

	Still	Walk	Run	Bike	Vehicle	Accuracy
F_{time}	93.73	82.21	97.47	67.75	87.51	84.79
F_{time}+F_{freq}	93.93	90.29	97.34	85.39	88.59	90.66
F'_{time}+F_{freq}	91.15	86.73	97.27	82.51	77.42	86.33

will be productive in improving mode-prediction accuracy. For cost, we want to make sure that a useful feature does not consume too much power to extract or generate. There are two sources of cost: power consumed by a signal source and power consumed by generating frequency-domain features via FFT. A gyroscope consumes 6mA versus the 0.1mA consumed by an accelerometer and 0.4mA by a magnetometer. We would select frequency-domain features and gyroscope signals only when they are proven to be productive.

We evaluated three sets of features:

- F_{time}: five time-domain features acc_std, acc_mean, mag_std, gyro_std and gyro_mean collected from accelerometer, magnetometer, and gyroscope.
- F'_{time}: three time-domain features acc_std, acc_mean, and mag_std collected from accelerometer and magnetometer.
- F_{freq}: two frequency-domain features acc_FFT (peak) and acc_FFT (ratio), generated from accelerometer's time-domain features.

Table 5 reports three sets of feature combinations. First, it indicates that including frequency-domain features yields improved accuracy. The second row of the table (F_{time} + F_{freq}) yields 6.5 percentile improvement over the first row (five time-domain features), especially in predicting the modes of walk and bike. This result demonstrates that the frequency-domain features are productive.

Since using a gyroscope consumes more than ten times the power consumed when using an accelerometer and a magnetometer, we evaluated the prediction degradation of removing the gyroscope. The third row of the table reports a lower accuracy than the first two rows. This result presents a dilemma: using the gyroscope is helpful but in order to conserve power, we should turn it off. Our solution to this dilemma is devising a virtual gyroscope scheme, which simulates physical gyroscope signals by an accelerometer and a magnetometer. We will report the good performance of the virtual gyroscope shortly in Section 5.3.2.

5.2.3 Performance of Classifiers

Table 6 reports and compares classification accuracy and confusion matrix for decision tree J48 (Weka default setting/with further pruning), AdaBoost (ten/three trees), and SVMs (RBF, degree-3 polynomial, and linear). The table shows that the SVM (degree-3 polynomial) enjoys a much higher accuracy 90.66% over the 84.81% of decision tree (Weka default setting) and 87.16% of AdaBoost (ten trees). (Notice that we have yet to factor in error correction.) Examining the three confusion matrices, SVMs perform more effectively in discerning between Walk and Bike, as well as Still and Vehicle.

We next examined the model size of our three candidate classifiers. Table 7 presents the test accuracy and model size of seven variations. We make three observations:

Table 6: Confusion table, number of instances, and test accuracy per class by using three selected classifiers. Each row represents the true mode, while each column represents the outputted mode. We average results of 10 runs, so the row sum in different tables may not be exactly the same because of numerical rounding.

	Still	Walk	Run	Bike	Vehicle	Accuracy
Still	5,136	141	1	7	1,847	72.01
Walk	309	9,595	45	698	270	87.89
Run	7	90	3,849	3	1	97.44
Bike	127	492	10	5,108	347	83.96
Vehicle	349	249	17	258	5,716	86.75

(a) Test accuracy and the confusion table using decision tree (Weka default setting). The average accuracy is 84.81.

	Still	Walk	Run	Bike	Vehicle	Accuracy
Still	5,507	60	0	4	1,560	77.23
Walk	317	9,795	39	580	187	89.71
Run	11	93	3,844	3	0	97.29
Bike	98	424	4	5,246	311	86.24
Vehicle	302	201	10	249	5,827	88.44

(b) Test accuracy and the confusion table using AdaBoost (10 trees). The average accuracy is 87.16.

	Still	Walk	Run	Bike	Vehicle	Accuracy
Still	6,699	22	0	2	409	93.93
Walk	466	9,859	25	450	119	90.29
Run	0	99	3,846	5	1	97.34
Bike	150	537	0	5,196	202	85.39
Vehicle	375	89	1	287	5,837	88.59

(c) Test accuracy and the confusion table using SVM (degree-3 polynomial). The average accuracy is 90.66.

- A simplified model reduces the model size and maintains competitive prediction accuracy. The further pruned decision-tree scheme saves 30% space and achieves virtually the same accuracy compared to the default decision-tree scheme.[4] Meanwhile, AdaBoost with three trees saves 70% space with a slightly lower accuracy.
- SVMs with kernels achieve a superior accuracy to both decision tree and AdaBoost. This result clearly indicates SVMs to be our choice.
- An SVM-classifier with a degree-3 polynomial is our choice not only because of its competitive accuracy, but also its remarkably small footprint.

We looked further into details of SVM kernel selection. Table 8 shows the result of using kernels of RBF and different polynomial degrees. As expected that SVMs with highly nonlinear data mappings (i.e., SVMs with the RBF kernel) performs the best. However, SVMs using polynomial expansions yield very competitive accuracy, only slightly lower than that of using RBF, but the model size can be significantly smaller. With the degree of polynomial mapping increasing gradually, the accuracy of the SVM classifier improves as well. After the degree is larger than three, the accuracy maintains at a level very close to RBF-kernel SVMs.

[4]Note that the study of [11] proposed a branch-and-bound algorithm to reduce decision tree's model nodes, which makes porting decision tree promising. However, decision tree with aggressive pruning cannot guarantee higher accuracy compared with the default setting.

Table 7: Test accuracy and model size using different classifiers.

Classifiers	Test accuracy		Model size
	No voting	Voting	
Decision Tree (default setting)	84.81	89.71	64.60KB
Decision Tree (further pruned)	85.55	90.04	45.71KB
AdaBoost (10 trees)	87.16	91.56	1,003.18KB
AdaBoost (3 trees)	85.89	91.09	246.92KB
SVM (RBF kernel)	91.53	94.10	1,047.97KB
SVM (degree-3 polynomial)	90.66	93.49	4.84KB
SVM (linear)	86.36	89.23	0.32KB

Table 8: Test accuracy without applying the voting scheme and model size of different SVM kernels.

Kernel of SVMs	Accuracy	Model size
SVM (linear)	86.36	0.32KB
SVM (degree-2 polynomial)	88.46	1.48KB
SVM (degree-3 polynomial)	90.66	4.84KB
SVM (degree-4 polynomial)	90.72	13.24KB
SVM (degree-5 polynomial)	90.73	31.72KB
SVM (degree-6 polynomial)	90.67	68.68KB
SVM (RBF)	91.53	1,047.97KB

Note that it is possible that polynomial-SVMs with low degrees perform slightly better than polynomial-SVMs with higher degrees, because a higher complexity model may suffer from a higher prediction variance causing overfitting.

In summary, we chose degree-3 polynomial as our SVM kernel for its competitive accuracy and extremely small memory and power consumption.

5.2.4 Evaluation of the Voting Scheme

In Section 3.3, we address the issue of short-term mode changes, and propose a voting scheme for correcting errors. Table 7 also compares the testing accuracy without and with the voting scheme. We can see that there is about a 4.5% enhancement for decision tree and AdaBoost. For SVMs, there is about a 3% enhancement after employing the voting scheme. The result shows that voting is effective for filtering out the short-term noise caused by e.g., mode-change, body movement, or poor road conditions.

5.3 Resource Conservation

We improve power consumption via one hardware and three software strategies. The hardware strategy—offloading computation from CPU to MCU or to our sensor hub—can clearly cut down power consumption. However, the hardware strategy alone is not sufficient as the 7.0mA power consumption can still be problematic for a wearable with a typical 315mAh battery. (A wearable runs several applications and cannot let the mode detector alone to drain its battery in 46 hours.) This subsection reports how we were able to reduce power consumption to 0.73mA, and the amount of power that each strategy can save.

5.3.1 Sensor Hub Saving

We implemented the transportation-mode classifier, SVMs with degree-3 polynomial, at two places of an HTC phone that is equipped with a sensor hub: one on the Android system platform and the other on the sensor hub platform (off the Android system). We then measured and compared their power consumption. The result shows a more than ten-folds power reduction by moving computation off the Android system to a sensor hub, from 88.5mA to 7.0mA. Notice that only 0.5mA is consumed by MCU, the rest 6.5mA is consumed by motion sensors.

Table 9: Performance of different SVM kernels on sensor hub. The total instruction cycles, processing time (ms) and power consumption (mA) are compared while running a 5-class classification (i.e., 10 decision functions for the one-against-one approach) using 7 features.

Kernel of SVMs	#Cycles	Time	Power
SVM (linear)	140	0.1	0.03
SVM (degree-2 polynomial)	720	0.6	0.15
SVM (degree-3 polynomial)	2,400	2.0	0.50
SVM (degree-4 polynomial)	6,600	5.5	1.38
SVM (degree-5 polynomial)	15,800	13.2	3.30
SVM (degree-6 polynomial)	34,320	28.6	7.15
SVM (RBF)	5458,000	4,548.33	1,137.08

To evaluate the performance of porting different SVM kernels on sensor hub, we estimated the power consumption and processing time according to the computational complexity of different SVM kernels. For polynomial kernel SVMs, the number of operation depends on the dimensionality of $\phi(\boldsymbol{x})$ after polynomial expansion. For example, if the degree for polynomial SVMs is 3 and the original feature number is 7, the dimension of $\phi(\boldsymbol{x})$ after polynomial expansion is 120. Therefore, $\boldsymbol{w}^T\phi(\boldsymbol{x}) + b$ takes 240 operations. Because 10 decision functions are evaluated for 5-class classification, totally 2,400 cycles are used for predicting an instance $\boldsymbol{x}$. For RBF-kernel SVMs, the decision value is determined by (8). To calculate $\alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) = \alpha_i y_i e^{-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}\|^2}$, the number of cycles is about $3n + 4$ (note that we can pre-calculate $\alpha_i y_i$ as one value and an exponential operation needs 2 cycles). In our 10-run experiments, the average number of support vectors for RBF kernel is $21,832$.

In detail, table 9 lists the number of instructions[5] which are used in each SVM kernel. In addition, the power consumption and total processing time are estimated according to the result of SVMs with degree-3 polynomial which we have measured. From this table, we can see that it is acceptable for only three SVM kernels to port to sensor hub, including the SVM (linear), the SVM (degree-2 polynomial), and the SVM (degree-3 polynomial). It is evident that the saving by moving computation to a sensor hub cannot be achieved by hardware alone, as we must shrink the footprint of the classifier to reduce the processing time, power consumption, and memory use.

5.3.2 Virtual Gyroscope Saving

Section 4.2 introduces our virtual gyroscope implementation. In this experiment, SVM (degree-3 polynomial) with 512 window size was used to evaluate power saved through our virtual gyroscope solution. The difference between using the physical gyroscope and using the virtual gyroscope is that the features of gyro_std and gyro_mean were replaced by data generated by an accelerometer to simulate gyro_std and gyro_mean.

The test accuracy and power consumption were compared between using a physical gyroscope and using our virtual

[5]ARM Cortex-M4 DSP assembler operates an add/subtract/multiply operation every one cycle, and an exponential operation every two cycles.

Table 10: Test accuracy and power consumption for resource conservation. The experiment runs SVM (degree-3 polynomial) in the sensor hub.

	Test accuracy		Power
	No voting	Voting	
Original setting	90.66	93.49	7.00mA
Virtual gyroscope solution	90.16	93.44	1.00mA
Virtual gyroscope (two-tier)	89.03	92.50	0.73mA

gyroscope. Table 10 shows that the virtual gyroscope can reduce 88% power, while maintaining relatively similar accuracy. Our recommendation is that on mobile phones where 7.0mA power consumption may not be an issue (7.0mA is relative low to a 2,500mAh battery), a physical gyroscope can be used. However, on wearables, virtual gyroscope is essential to reduce the power consumption to 1.0mA so that a 300mA battery can last for at least a couple of days.

5.3.3 Two-Tier Decision Saving

Our two-tier scheme wakes up the mode detector only when the accelerometer detects $\mathsf{acc\_std} \geq 0.06$. From Table 1 we can see that the accelerometer consumes 0.1mA and MCU also 0.1mA when in the idle mode. Thus, the sensor hub consumes only 0.2mA when it is in the idle mode.

If we assume that $\mathsf{acc\_std}$ is lower than the threshold in a third of a day, the average power consumption per hour is

$$\{0.2\ (\text{mA}) \times 8\ (\text{hr}) + 1.0\ (\text{mA}) \times 16\ (\text{hr})\}/24\ (\text{hr}) \cong 0.73\text{mA}.$$

Regarding mode-prediction accuracy, the result shows that the test accuracy becomes slightly lower but the power consumption is reduced by 27%.

6. CONCLUSION AND FUTURE WORK

This study presents our hardware-software co-design of a classifier for mobile and wearable devices to detect a person's transportation mode (i.e., in mode of still, walking, running, biking, or on a vehicle). Particularly, we focus on addressing the big-data small-footprint requirement by designing a classifier that is low in both computational complexity and memory requirement. Together with a sensor-hub configuration, we are able to drastically reduce power consumption by 99%, while maintaining 92.5% accuracy in detecting five transportation modes.

The power saving we achieved and hence our contributions can be summarized as follows:

1. *Classifier footprint reduction.* Using the polynomial degree-3 kernel significantly reduces both computational complexity and memory requirement for our classifier from $\mathcal{O}(ln)$ to $\mathcal{O}(\binom{n+d}{d})$, where in general the latter is much smaller than the former according to (7). This reduction permits the classifier to fit into the memory of the sensor hub as well as reduces power consumption. Together with an MCU that can run at a reduced rate of 16MHz, we reduced power consumption from 88.5mA to 7.0mA. (If the footprint is large, the MCU must run at 72Hz to maintain short latency, and the power consumption shoots up several times.)
2. *Virtual gyroscope design.* Since this 7.0mA power usage is still not acceptable by wearables, we designed and implemented a virtual gyroscope to cut down power consumption further to 1.0mA.
3. *Two-tier decision.* Though further saving is minor (from 1.0 to 0.73mA), we showed that a simple admission control scheme can reduce power by 27%.

Our implementation was launched with the HTC One (M8) model world-wide on March 25^{th}, 2014.

Our future work includes three extensions to improve the classifier's adaptability and scalability.

- *Multi-tier extension.* With merely information from three sensors, we are able to achieve 92.5% mode-prediction accuracy. As mentioned in the beginning of Section 3, when side information is available on e.g., the cloud server, where resources are virtually unlimited, accuracy can be further enhanced. For instance, when transit route information is available, telling between the modes of driving a car and taking a bus is much simplier. We plan to provide multi-tier extensions when a target application requires higher acuracy and when both side information and resources are available.
- *Data cleasing.* One factor hindering higher detection accuracy is signal noise. While the environment may introduce noise, human factor contributes significant interferences. For instance, a jogger who holds her phone steadily to select a song may be detedted to be in the still mode. A person playing a game on his phone when sitting may be regarded as running. Although a voting procedure has been implemented in our system to remove outliers, more advanced techniques are expected to help further reduce the effects of various noise characteristics.
- *Generalization validation and enhancement.* In this study, we collected transportation-mode data in one country, Taiwan. We now begin to evaluate the accuracy of the classifier when being used in other countries where people are of different builds and roads/vehicles of different conditions. We regard such as an important big-data learning problem (or in general, a machine learning problem) where the unseen data (testing data) and the training data may exhibit different statistical characteristics. Traditional machine learning tasks assume that the testing data and the training data observe the same statistical distribution. We will investigate the gereralization capability of our mode-detection classifier when it is applied to classify unseen data of a slightly different distribution, and devise remedies if adaptation is required.

7. ACKNOWLEDGMENTS

We would like to thank Zih-Ci Lin, Stanley Wu, and Cooper Lin for their support, and also thank Alex Wu, Brian Mei, Relic Chu, and Dave Wu for their helpful suggestions.

8. REFERENCES

[1] adidas. miCoach SPEED CELL. `http://www.adidas.com`.

[2] I. Anderson and H. Muller. Practical activity recognition using GSM data. Technical Report CSTR-06-016, Department of Computer Science, University of Bristol, July 2006.

[3] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[4] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. A. Müller,

E. Säckinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwriting digit recognition. In *International Conference on Pattern Recognition*, pages 77–87, 1994.

[5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] E. Y. Chang. *Foundations of Large-Scale Multimedia Information Management and Retrieval*. Springer-Verlag New York Inc, New York, 2011.

[7] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

[8] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

[9] Fitbit. Flex wristband. `http://www.fitbit.com`.

[10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[11] M. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. Building decision trees with constraints. *Data Mining and Knowledge Discovery*, 7(2):187–214, 2003.

[12] K.-S. Goh, E. Chang, and K.-T. Cheng. Svm binary classifier ensembles for image classification. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM)*, pages 395–402, 2001.

[13] Google. Google now. `http://www.google.com/landing/now/`.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11, 2009.

[15] HTC. HTC Research. `http://research.htc.com/2014/06/publication14001/`.

[16] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing: Algorithms, Architectures and Applications*, 1990.

[17] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Lecture Notes in Computer Science*, volume 3096. Springer, 2006.

[18] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 71–84, 2010.

[19] Monsoon Solution Inc. Power monitor. `http://www.msoon.com/`.

[20] M. Y. Mun, D. Estrin, J. Burke, and M. Hansen. Parsimonious mobility classification using GSM and WiFi traces. In *Proceedings of the Fifth Workshop on Embedded Networked Sensors (HotEmNets)*, 2008.

[21] S. Nath. ACE: exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 29–42, 2012.

[22] Nike. Fuelband. `http://www.nike.com/us/en_us/c/nikeplus-fuelband`.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[24] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. SociableSense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, pages 73–84, 2011.

[25] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks*, 6(2):13:1–13:27, 2010.

[26] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. Smith, S. Consolvo, and W. Griswold. Mobility detection using everyday GSM traces. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, 2006.

[27] D. Specifications. Sony smartwatch 2-battery. `http://www.devicespecifications.com/en/model-battery/518829ce`.

[28] V. Srinivasan and T. Phan. An accurate two-tier classifier for efficient duty-cycling of smartphone activity recognition systems. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones (PhoneSense)*, pages 11:1–11:5, 2012.

[29] M. E. Stanley. Building a virtual gyro. `https://community.freescale.com/community/the-embedded-beat/blog/2013/03/12/building-a-virtual-gyro`, 2013.

[30] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu. Transportation mode detection using mobile phones and GIS information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, 2011.

[31] S. Wang, C. Chen, and J. Ma. Accelerometer based transportation mode recognition on mobile phones. In *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems*, pages 44–46, 2010.

[32] P. Widhalm, P. Nitsche, and N. Brandie. Transport mode detection with realistic smartphone sensor data. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, pages 573–576, 2012.

[33] G. Wu and E. Y. Chang. KBA: kernel boundary alignment considering imbalanced data distribution. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):786–795, 2005.

[34] J. Yang. Toward physical activity diary: motion recognition using simple acceleration features with mobile phones. In *Proceedings of the 1st international workshop on Interactive multimedia for consumer electronics*, pages 1–10, 2009.

[35] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on gps data. In *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp)*, pages 312–321, New York, NY, USA, 2008. ACM.