WHI REU 2013

Energy Management Based on Charging Behavior Prediction

Networked & Embedded Systems Laboratory, UCLA

Mikhael Semaan

Prashanth Swaminathan
Chenguang Shen
Kevin Ting

Mani Srivastava

ABSTRACT

As mobile device applications continue to include richer content, the need for maximizing power efficiency on the devices increases. While a great deal of effort is put into designing more energy efficient hardware platforms, not much attention has been given to individual users' charging behaviors.

Assuming that every user will charge his/her device differently, we can potentially increase battery life by dynamically scheduling tasks to take place when their impact on the battery is minimal (such as during an overnight charge). In order to allow this optimization, we made use of machine learning algorithms to predict charging behavior.

We gathered data using SystemSens, an Android program developed by Dr. Hossein Falaki. Through testing and cross-validation in the machine learning suite Weka, we found that the "RandomTree" classifier yielded the most accurate results in tandem with our feature selection. We used the raw reported values of battery status, battery level, and time, as well as extracted features for the last charging time, last charging level, and duration of last charge. These proved to be the most important features for predicting the time before the next expected charge, giving us approximately 90% accuracy on our training sets.

After finalizing our feature and classifier selection, we developed an architecture in which the phone locally polls data, uploads it to a server for training a model, and then receives that model and runs it on the phone, providing a platform for accurately predicting individual charging behavior.

BACKGROUND/IMPACT

The market for mobile computing devices -- smartphones, tablets, small form-factor notebooks -- has been rapidly expanding. Devices like smartphones are particularly attractive to many users because they can serve as all-in-one media and communication centers: they allow for keeping contact over talk, text, e-mail, social media; and they are capable of internet browsing, multimedia playback, online banking, and a host of other things all on a single device.

A relatively recent potential application for smartphones involves work in the area of wireless health. There are a host of novel ways to use smartphones as a central communication hub for external sensors, or for inferring information from the phone's own sensors, and many of them involve constant communication with an external server for processing. This idea opens up many possibilities in terms of real-time monitoring of health data and the landscape of health care. However, there is one major issue: modern smartphones and other mobile devices, while very powerful, are limited by battery life.

While there is considerable effort being put into designing power-efficient hardware platforms for mobile devices, not much attention has been given to the individual charging habits of different users. Ideally, based on a user's charging behavior, there should be a way to dynamically schedule power-intensive tasks on the device for when they will have minimal battery impact, or hold off non-crucial tasks when they may significantly affect battery performance.

RESEARCH QUESTION/SPECIFIC GOALS

The primary question we investigate is: How can we reliably predict a user's charging behavior? By answering this, we can potentially open the door for continued work in user-tailored energy management, and we intend to address this in three major increments.

The first goal of this work is to collect data about users and their charging behavior. We begin with the assumption that there will be some regularity to an individual's behavior, but the preliminary data collection should provide clues about whether or not this assumption is valid. Once we have collected sufficient data, we want to try to use that data to predict an individual's charging behavior; we intend to address this with machine learning algorithms. If we can obtain a reliable model for predicting charging behavior, the final aim of the project is to implement that model into an application-server framework that is sufficiently low-cost to the device.

METHODOLOGY/GENERAL APPROACH

Data Collection

The first branch of this project involved collecting data on smartphones and other mobile devices. For this, we used a modified version of SystemSens, an Android application developed by Dr. Hossein Falaki out of UCLA's Center for Embedded Network Sensing (CENS). [1] SystemSens polls every available sensor on the phone, packages the different kinds of data into unique JSON objects, and sends the data to an external server. We modified the SystemSens code to prevent it from sending potentially sensitive data such as GPS and network location data, and to send the data to our own internal server in the lab where we can access and process the data.

Initially, we installed our modified SystemSens on two lab phones and began collecting data from these phones. During the collection period, we wrote MATLAB scripts to collect the data from the server, parse the JSON objects, and sort the data into its categories. However, as the datasets grew in size, we encountered heap space errors in MATLAB, and around this time we also knew that we wanted to use the machine learning suite Weka [2] to do the modeling and classification. Thus, we converted our existing MATLAB scripts to Python and added a layer to package the data into Weka's ARFF format. In addition, based on our initial results and the amount of data being collected, we decided to begin classification only using time and battery data from SystemSens (voltage, current, level, health, temperature, charging state, timestamp).

Machine Learning in Weka

As mentioned previously, we used Weka to do our machine learning work. The major question in this part of the project asks whether we can, in fact, accurately predict a user's charging behavior; and if we can, which combinations of features and classifiers give us the best solution.

To answer this question, we ran extensive testing on our datasets -- both in the preliminary stages and as the data grew in size -- and on different feature selections and classifiers. In addition to the raw data of timestamp, voltage, current, level, health, temperature, and charging state, we included features for the time of week, time of last charge, length of last charge, and last charging level; and we wanted to predict the time until the next charge. We also tested on a number of classifiers, including nearest-neighbor algorithms such as IBk and decision trees including J48, DecisionStump, REPTree, RandomForest, and RandomTree.

Eventually, we found that RandomTree gave us the best results consistently and across our datasets, along with the time of week, level, charging state, time of last charge, length of last charge, and last charging level features. The RandomTree decision tree works by randomly selecting *k* attributes for classification, and splits on that set of attributes before reiterating on a new random selection, and so on. In order to use this model, we converted the numeric value for time until next charge to a nominal value, where each nominal increment represents fifteen minutes of time. Additionally, the lazy.IBk classifier gave us nearly identical performance for our later datasets, but did not show the same consistency early on.

Application-Server Framework

After deciding on an optimal classifier and feature combination, we proceeded to write an application that implements the machine learning algorithm on the phone and runs the model locally. Initially, we were interested to see what kind of computational power the process of training and building a model took. As it turned out, the model construction is extremely power-hungry, and is therefore impractical to run on the phone locally, so we built a framework between an external server and the phone. Here, the phone application polls the relevant SystemSens data, and occasionally connects to the server (currently this happens nightly) to dump the data and request a new model. On the server side, we implemented the power-hungry model building, and then send back the completed model to the phone's application, which runs the pre-built model locally. This setup minimizes the battery impact on the phone because running the completed model takes very little power and can be done quite often, while the power-hungry training task is offloaded to an external server, and is only done sparsely and over a short period of time.

INITIAL DATA AND RESULTS

As we noted earlier, RandomTree ultimately gave us the best classification results with our selection of features. The following tables summarize some of the results from our later datasets on test subjects 1 and 2 using 10-fold cross-validation, and includes the Weka-reported time to build the model (NOTE: this does not accurately reflect how long the training took, as some models took significantly longer to evaluate. DecisionStump completed relatively quickly, but yielded horrible accuracy; while lazy.IBk took almost as long as the other algorithms despite the very small reported time.)

[FIGURE 1]

The lazy.IBk classifier's performance in the late datasets was surprising -- and is indeed an area worth investigating as an alternative. However, due to its much larger variance with our early datasets, and because in the application we want to emphasize consistently reliable performance, we have opted to continue with the RandomTree implementation, though we do feel that IBk is worth looking into.

DISCUSSION

From the results, the most interesting thing worth noting is about IBk's last-minute performance surprise. Earlier during our testing, RandomTree handily beat every other classifier, including lazy.IBk, so we proceeded to continue our work with RandomTree. The results which show IBk taking an admittedly miniscule (but significant, because it was not near it before) lead in accuracy are based on very late testing.

This is an area that we cannot investigate further due to time constraints, but one which presents a very interesting choice. RandomTree is sufficiently lightweight and fast on the phone, while we cannot say whether IBk is, since we have no further testing in the area. The Weka-reported size is many times smaller than even RandomTree, though, and the actual speed (not the Weka-reported time to build model) of the classifier is at least on par.

We suspect that RandomTree is still the better choice overall, since it gave us much better consistency in the earlier and more varied datasets. However, again, this is an area we feel is worth investigating.

CONCLUSION/FURTHER WORK

Overall, our conclusions seem to be that we can, in fact, reliably model charging behavior, and that implementing machine learning on a mobile device is practical with some smart management of server-

phone resources. This result is promising for the future work it enables: namely, dynamic task management based on these models to maximize battery performance. This is a very potentially useful idea, since it can enable significantly better battery performance and the ability to continue adding feature-rich content to the devices (such as wireless health applications) without the expense of hardware-level changes. Thus, we have shown the promise of machine learning in modeling individual behavior, implemented this infrastructure on a smartphone, and opened the door for future work in battery optimization and continued support of fully featured content on mobile devices.

REFERENCES

[1] Falaki, H. and Mahajan, R. and Estrin, D. SystemSens: A Tool for Monitoring Usage in Smartphone Research Deployments, Proceedings of the sixth ACM international workshop on mobility in the evolving internet architecture, June 28-28, 2011, Bethesda, Maryland, USA

[2] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

APPENDIX

[FIGURE 1]

Test Subject 1:

BEST FEATURES

Classifier	Model Size (MB)	Time to Build Model (seconds)	% Correctly Classified
lazy.IBk	1.30	0.01	91.57%
trees.J48	66.40	3.47	86.99%
trees.DecisionStump	0.06	0.26	5.76%
trees.REPTree	62.20	2.11	73.47%
trees.RandomTree	21.70	1.04	91.44%

RAW DATA

Classifier	Model Size (MB)	Time to Build Model (seconds)	% Correctly Classified
lazy.IBk	1.10	0.01	75.09%
trees.J48	70.20	3.06	80.43%
trees.DecisionStump	0.06	0.35	5.76%
trees.REPTree	62.60	1.76	71.53%

trees.RandomTree	31.30	1.80	85.88%

Test Subject 2:

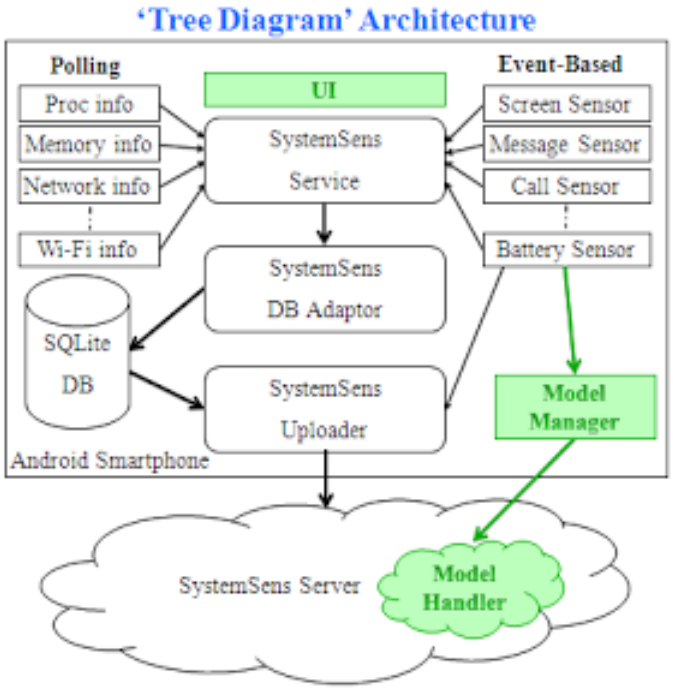
BEST FEATURES

Classifier	Model Size (MB)	Time to Build Model (seconds)	% Correctly Classified
lazy.IBk	1.60	0.00	94.67%
trees.J48	39.40	3.31	92.31%
trees.DecisionStump	0.06	0.36	4.86%
trees.REPTree	56.00	2.35	84.17%
trees.RandomTree	17.50	1.27	94.64%

RAW DATA

Classifier	Model Size (MB)	Time to Build Model (seconds)	% Correctly Classified
lazy.IBk	1.30	0.04	80.93%
trees.J48	56.80	3.22	88.97%
trees.DecisionStump	0.06	0.43	4.88%
trees.REPTree	58.00	1.46	84.94%
trees.RandomTree	24.10	1.51	90.69%

[FIGURE 2]


'Tree Diagram' Architecture
Polling
Event-Based
UI
Proc info
Memory info
Network info
Wi-Fi info
SystemSens
Service
Screen Sensor
Message Sensor
Call Sensor
Battery Sensor
SystemSens
DB Adaptor
SQLite
DB
SystemSens
Uploader
Model
Manager
Android Smartphone
SystemSens Server
Model
Handler