A Model for Hour-Wise Prediction of Mobile Device Energy Availability

47

Mathias Longo, Cristian Mateos, and Alejandro Zunino

Abstract

Mobile devices have become so ubiquitous and their computational capabilities have increased so much that they have been deemed as first-class resource providers in modern computational paradigms. Particularly, novel Mobile Cloud Computing paradigms such as Dew Computing promote offloading heavy computations to nearby mobile devices. Not only this requires to produce resource allocators to take advantage of device resources, but also mechanisms to quantify current and future energy availability in target devices. We propose a model to produce hour-wise estimations of battery availability by inspecting past device owner's activity and relevant device state variables. The model includes a feature extraction approach to obtain representative features/variables, and a prediction approach, based on regression models and machine learning classifiers. Comparisons against a relevant related work in terms of the Mean Squared Error metric shows that our method provides more accurate battery availability predictions in the order of several hours ahead.

Keywords

Mobile cloud computing · Battery prediction · Feature selection · Time series · Android

47.1 Introduction

Mobile Cloud Computing (MCC) [1] is a computing paradigm that has been proposed as a way of augmenting mobile devices –and hence deal with their inherent limitations– with remote resources located in the Cloud. To this end, MCC combines advances from the areas of distributed architectures, mobile computing, cloud computing and wireless/fixed networks so that rich applications can be seamlessly and efficiently "executed" in mobile devices via the actual execution/processing of computations/data on remote Cloud resources. Examples of such applications are speech recognition and augmented reality. Moreover, moving computations and data from devices to remote resources while obtaining maximal benefit from the system in terms of application execution performance and energy saving is performed via offloading techniques [1–3].

Traditional MCC proved however insufficient to cope with many latency-sensitive applications and very large number of (mobile) client devices, such as critical IoT (Internet of Things) services [4]. In response, the Fog Computing paradigm [5] was proposed around 2012 by Cisco researchers to provide highly-scalable infrastructures for developing and deploying latency and location-aware applications, where geographical distribution, mobility and software/hardware heterogeneity are the rule. Fog Computing has the ability of augmenting mobile (e.g., laptops, smartphones, tablets, wearables) and wireless devices (e.g., sensors) with nearby fixed processing/storage resources. Moreover, motivated by the huge amount of mobile devices –there will be 1.5 mobile-connected devices per capita by 2021 (http://tinyurl.com/mokcut3)– with ever-growing software and hardware capabilities, recent research [6] has suggested the possibility of using nearby mobile devices as destination for offloading computations/data, paradigm which has been named Dew Computing [7, 8].

Among the many challenges this new computing paradigm intuitively poses is resource scheduling/allocation, i.e. the mechanism by which tasks or data to be run or processed, respectively, are placed in appropriate computational resources (mobile devices in this case). Compared to resource scheduling in conventional distributed

M. Longo
University of Southern California, Los Angeles, CA, USA
e-mail: mathiasl@usc.edu

C. Mateos (✉) · A. Zunino
ISISTAN-UNCPBA-CONICET, Tandil, Buenos Aires, Argentina
e-mail: cristian.mateos@isistan.unicen.edu.ar;
alejandro.zunino@isistan.unicen.edu.ar

S. Latifi (ed.), *Information Technology – New Generations*, Advances in Intelligent Systems and Computing 738,
https://doi.org/10.1007/978-3-319-77028-4_47

computing environments, resource scheduling using a cluster of nearby mobile devices is challenging due to the highly dynamic and heterogeneous nature of resource availability [9]: mobile devices can change their location, they are non-dedicated by nature and have energy limitations. Quantifying resource availability, on the other hand, is of utmost importance for a Dew Computing scheduler to take good scheduling decisions, i.e. selecting the most appropriate nearby devices to execute some given computations. However, such quantification is difficult in light of Dew Computing high dynamism and heterogeneity.

In this context, we investigate in this paper a model to quantify resource availability in mobile devices that take into account the usage profile of device owner's, which is an open problem seldom explored in the literature [10]. The model produces ahead predictions of remaining battery in the order of hours, and it is built by analyzing relevant past owner's activity (charging state, application execution, activated radios, etc.). Both the model itself and the assessment of its predictive accuracy are based on real traces from the Device Analyzer data-set [11], which is at present the largest mobile device usage data-set and includes traces from more than 31,000 users worldwide and different Android-based mobile device brands.

The organization of this paper is as follows. The next section explain our model to predict energy availability in detail. Then, Sect. 47.3 evaluates the accuracy of the model in terms of the Mean Square Root Error metric. Finally, Sect. 47.4 summarizes the implications of our findings and delineates future works.

47.2 Approach

In this section, we present our approach for predicting mobile device battery lifetime based on owner's activity, i.e. device usage patterns. Based on the activity traces of an individual user, our approach relies on feature selection to select the most relevant activity types for building a prediction model. This is explained in detail in Sect. 47.2.1. Feature selection pursues two goals, namely reducing the number of features to reduce overfitting and improve the generalization of models, and gaining better understanding of the features and their relationship to the response variables [12]. Moreover, we also propose an ensemble of several Machine Learning algorithms, trained using the resulting features from previous steps, which is detailed in Sect. 47.2.2. For the sake of illustrating the proposal, values shown below correspond to applying the model to a user from the data-set, who has registered activity traces for over 6 weeks.

Lastly, the implementation of this approach, as well as the creation of the Machine Learning models, was done in Python using open-source libraries: Scikit-Learn, Pandas, Numpy and MatplotLib. The source code is available from a GitHub repository.[1]

47.2.1 Preliminary Data Analysis and Feature Selection

To come up with an approach to feature selection that is general enough in the problem domain at hand, we have as mentioned earlier focused our research in the Device Analyzer data-set.

Before analysing the data, we defined a structured format for it. To this end, the data-set that basically consists in plain logs is splitted into states, where each state has the value for each mobile sensor in a device. A mobile device state changes as a result of an event in time that changes the value of any sensor. For example, a change in the battery level can be defined as an event, which triggers a new state of the mobile device in which the battery level is modified and all other features remain the same. Initially, from the formatted data, the following combination of features were considered:

- Day of week: Day of week where the event occurred, numbered from 0 to 6. Type: Integer.
- Minute: Minute of the day in which the event took place. Type: Integer.
- External Supply: Whether the mobile device is plugged to an external energy supply, e.g. AC adapter or USB connection. Type: Boolean.
- Bright Level: Screen brightness intensity. Type: Integer.
- Screen on/off: Whether the mobile screen is active or inactive. Type: Boolean.
- Connected: Whether the mobile phone is connected to a 3G/4G network. Type: Boolean.
- Connected to Wifi: Whether the mobile phone is connected to a Wifi network or not. Type: Boolean.
- Temperature: Mobile phone's actual temperature. Type: Integer.
- Voltage: Current battery voltage. Type: Integer.
- Battery Level: Current battery level ranging between 0 and 100. Type: Integer.

The dataset itself contains several other features, such as those related to location and application usage. Nevertheless, these features were not considered as relevant to the battery level modelling as the ones previously listed. First of all, it is true that by knowing the location of a user in each state, it might be possible to infer if they are at home. If that is the case, the probability of charging the mobile phone increases.

[1] http://github.com/matlongo/battery-level-predictor.

Table 47.1 Extract from the formatted data-set for an individual user

Day of week	Minute	External supply	Bright level	Screen on/off	Connected	Connected to Wifi	Temperature	Voltage (mV)	Battery level
3	652	0	149	1	0	1	275	3686	31
3	653	0	149	1	0	1	286	3740	30
3	654	0	149	1	0	1	286	3740	30
3	655	0	149	1	0	1	286	3740	30

Cells in Boolean-typed columns contain either 0 (false) or 1 (true)

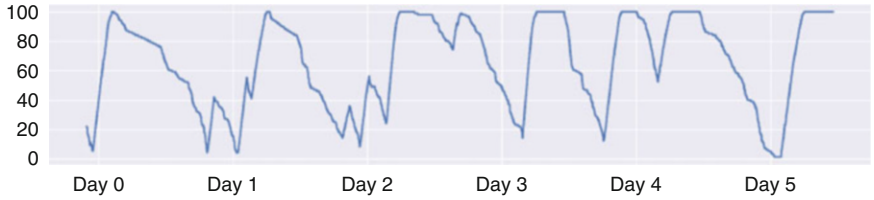

Fig. 47.1 Battery level overview of the sample user in the first 5 days

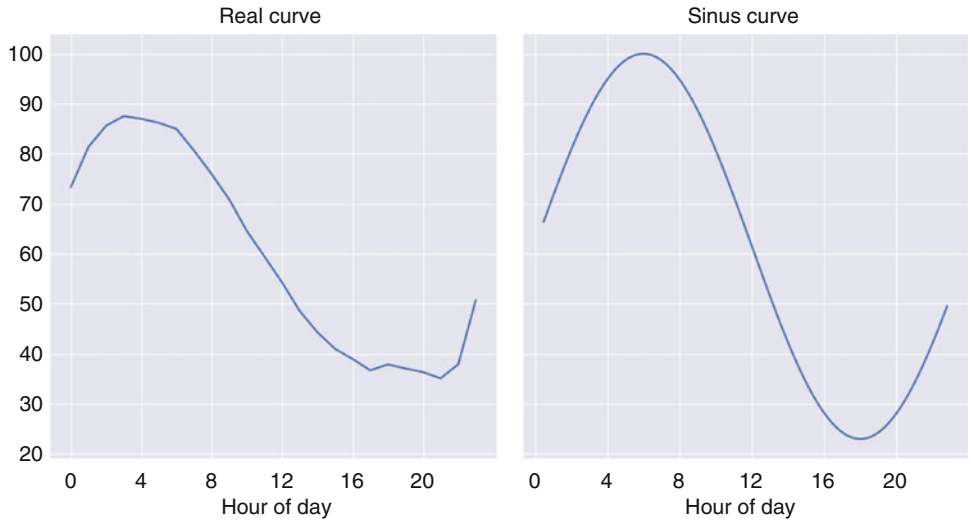

Fig. 47.2 Per-hour average battery level of the sample user: real (left) and sinus curve (right)

However, that information can also be obtained from the charging pattern using the Energy Supply feature. Secondly, application usage is highly seasonal, because applications tend to change in a short time, as well as the users might also change their applications periodically (e.g. replacements). Therefore, it is very difficult to generalize a model based on this feature, and it might incur in extra noise.

Table 47.1 shows an extract from the formatted data-set for one particular mobile device user. Furthermore, Fig. 47.1 depicts the battery level variation along time for this user (first 5 days of the sample). It is possible to see that the resulting curve is not exactly periodic since it takes different shapes, and it does not have a preset behavior. However, it is worth noting that there is a visual resemblance to a sinusoidal curve, because it continuously goes up and down. In fact, Fig. 47.2 (left) depicts a greater resemblance when the battery level is averaged per day. Considering this observation, which applied to many other users in the data-set as well, a new feature representing the sinus movement was added to the feature list: $Battery\,sinus = amplitude * sin(2 * pi * \frac{minute}{minsPerDay}) + batteryLevelMean$, where *amplitude* describes how much the curve goes up and down, *minute* is the minute of the day, *minsPerDay* is the total number of minutes in a day (i.e., 1440), and *batteryLevelMean* is the average battery level per day. The *batteryLevelMean* is the mean per day battery level in the dataset. In addition, the maximum *amplitude* is 50, since a bigger value makes the curve going above the maximum battery level or below the minimum battery level. Finally, it is worth pointing out that the only variable used to calculate the sinus is *minute*, which is easily accessible in every mobile device. Figure 47.2 (right) shows the resulting curve compared to the real one (in the left) we previously obtained using real battery level samples.

Likewise, from Fig. 47.1 it is possible to see that the battery level tends to go up during the night, due to the fact that the energy supply is connected. Therefore we can model the energy supply behavior with a cosine, getting the zenith during the night when it is usually connected, and going down during daytime. Concretely, we included the following extra feature: $Battery\,cosine = amplitude * cos(2 * pi * \frac{minute}{minsPerDay}) + batteryExternalMean$. In this case the *amplitude* has a maximum value of 0.5, because the

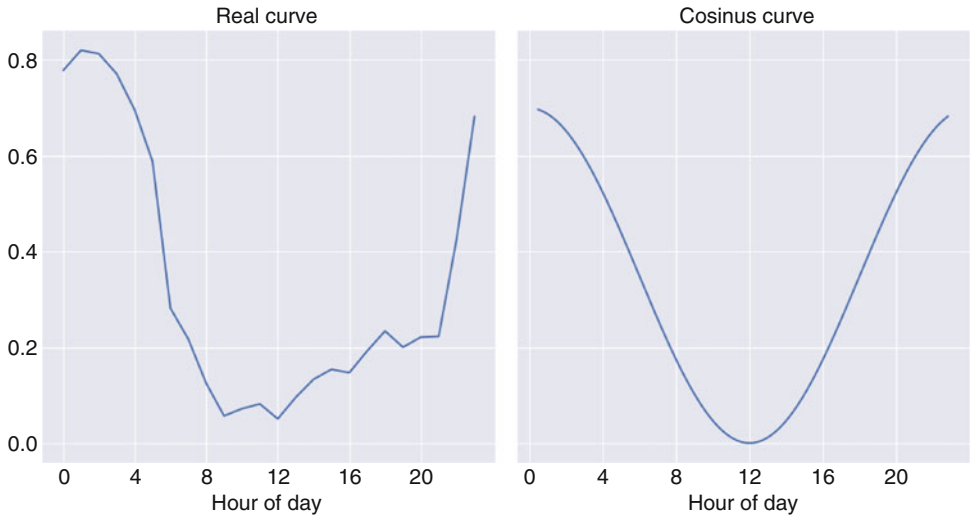

Fig. 47.3 Per-hour average energy supply feature value on sample user dataset: real (left) and cosine curve (right)

energy supply can only take values 0 or 1. Then, *batteryExternalMean* is the average of all the external supply feature values in the dataset. Figure 47.3 (right) shows the curve obtained from this calculation, and Fig. 47.3 (left) shows the real averaged curve.

Table 47.2 Dickey-Fuller test: results

Test statistic	−13.37
Critical value (5%)	−2.86
Critical value (1%)	−3.43

47.2.1.1 Correlation Analysis

Based on the features included in the data representing any mobile user activity with his/her device, we performed a correlation analysis between the features established in previous steps, using the Pearson correlation coefficient.

First of all, there is a very strong positive correlation between battery level and voltage. This is due to the fact that the voltage is used as a variable to carry out the battery level. This means that voltage is not discriminant for our model and it will overfit any Machine Learning model, hence it should not be considered for the following steps. In addition to the voltage, the temperature should also be dismissed for future analysis, because it has a remarkable negative correlation with the battery level.

Secondly, we also obtained a very strong negative correlation between *Connected* and *Connected to Wifi*. That means, whenever the Wifi is connected, 3G/4G is not used, and vice versa. This might be because of the way Android works, since Android turns off 3G/4G connection when there is Wifi available so as to avoid incurring in monetary costs or wasting mobile data quota.

A more interesting relationship to point out is between the sinus, one of the last added features, and the battery level. This is a good symptom that the model can rely on this feature to predict the battery level for future states. Although slightly weaker, there is a noticeable positive correlation between the External Supply and the cosine which was also added as a new feature.

Another interesting point is that the screen tends to be off when the external supply is connected. Probably, the mobile device was left charging without so much interaction with it. Besides, there is a negative correlation between the hour (in minutes) and the battery level, meaning that as time passes by the mobile device tends to have less battery level. This behavior is consistent with the curve depicted in Fig. 47.1.

47.2.1.2 Time Series Analysis

Given the fact that the input data-set is a collection of data points gathered at constant time intervals -established by the Device Analyzer Android app that collects samples in a device- and that the battery level is time dependent, Time Series analysis can be used to determine how many *lag* intervals should be used to estimate a future battery level. For the purpose of this work, a *lag* is the number of previous states that are needed to come up with a good battery level estimation. Since intuitively estimations have a strong dependence on previous state's values, it is necessary to know how many previous states should be considered for the prediction.

We first checked if the time series is stationary. A time series is said to be stationary if its statistical properties such as mean and variance remain constant over time. Intuitively, it can be inferred that if a time series has a particular behavior over time, there is a very high probability that it will follow the same in the future. More formally, stationarity can be checked by using the Dickey-Fuller Test [13]. In this case, the null hypothesis is that a time series is non-stationary. Depending on the resulting critical value from the test, the level of confidence to say that the time series is stationary might be higher or lower. Table 47.2 shows the results for

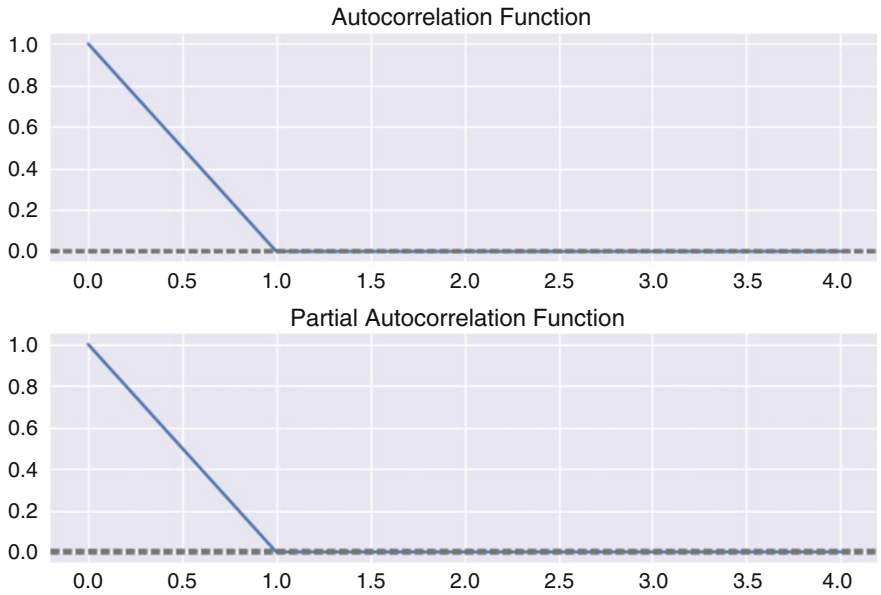

Fig. 47.4 Autocorrelation and Partial Autocorrelation for battery level feature along the time

the Dickey-Fuller Test on the data-set. As we can see, the statistic test is less than the 1% critical value, meaning that there is a 99% of confidence to say that the time series is stationary.

Once proven its stationarity, we decided to apply an Auto-Correlation Function (ACF) and a Partial Auto-Correlation Function (PACF) to determine the number of lags that best represent the following state. Figure 47.4 shows the ACF and PACF applied to the dataset. It is possible to visually infer, according to the figure, that one lag is sufficient to estimate the following state. Therefore, a new feature is added to accomplish this relationship: Previous Battery Level.

Table 47.3 Feature selection using F-regression, mutual information and Lasso: scores

	F-Regression	Mutual Information	Lasso
Previous battery level	3.86E+09	3.995	0.9966
Minute	2.06E+05	0.599	0.0088
Day of week	5.25E+02	0.067	0.0002
External supply	1.28E+05	0.193	0.467
Connected	1.21E+03	0.028	0
Connected to Wifi	2.85E+03	0.024	0
Bright level	2.87E+04	0.002	0.0002
Screen on/off	3.22E+04	0.059	0.007
Sinus	2.61E+05	0.308	0.086
Cosine	6.39E+03	0.244	0

47.2.1.3 Feature Selection

Once the dataset was preprocessed and the feature extraction steps is finished, we performed feature selection to obtain the minimal set of descriptive features. In this part of the process those features that are not influential in the battery level are filtered out. There exist many strategies that can be adopted for this part of the pipeline [12], and then, in this case, a combination of three different approaches have been considered.

First of all, two *Univariate Feature Selection* methods were considered: F-Test and Mutual Information. The former one is a statistical test for estimating the degree of linear dependency between two random variables, so it is performed for each feature against the battery level. On the other hand, the latter one is a non-parametric method that uses entropy to measure information content in data, in the sense that the higher the entropy the lower the information. Finally, the third approach considered is Lasso regression. Lasso regression is a regression model with $L_1 = |w_i|$, regularization term, where w_i is the coefficient used for x_i. Compared to the Ridge Regression regularization term, Lasso shrinks those "irrelevant" coefficients to zero instead of shrinking them to a small value. This model is parametrized by a meta-parameter called *alpha*. The larger the value of *alpha* the fewer the features selected.

Table 47.3 depicts the values obtained from the three mentioned techniques. The first column shows the F-Score calculated from the correlation between each feature and the target feature, ie, the battery level. The second column depicts the mutual information score calculated from the entropy of the dataset. Lastly, Lasso scores are the resultant coefficients associated for each feature after training the regression model with the regularization term.

47.2.2 Model Construction

In this section, to explain the proposed model to predict mobile phones battery level in future states. In particular, it is a result of combining classification models and a regression model to predict a new state based on a previous one. First of all, the selected model to estimate the battery level is a Multiple Linear Regression Model built using the resulting features from previous section. Multiple Linear Regression models are the most suitable Machine Learning algorithms for predicting a continuous variable, such as mobile phones battery level, depending on a set of different features. A Time Series model is not enough for this particular case, because it dismisses all the non-time related features. Moreover, the model is trained using the first 2 weeks of gathered data for a user, and the rest of the data-set is used for testing purposes.

In addition to the regression model, two complementary classification models were built to estimate the variability of external supply and screen on/off features. These models predict the state of both features in future states. The external supply classifier is a Random Forest classifier with ten estimators and its maximum depth is 3. It was trained using the cosine as well as the minute of the day that is being predicted and the previous state. The screen on/off estimator is a Decision Tree classifier built using the minute of day and the previous screen state.

The proposed mechanism to combine all the created models is an iterative method that estimates the following state based on the previous estimated states. The following is the pseudocode of the proposed method:

```
while current_state < desired_state:
  current_state[external_supply] =
    external_supply_classifier.estimate(current_state)
  current_state[screen] =
    screen_classifier.estimate(current_state)
  current_state =
    regression_model.estimate_following_state
    (current_state)
```

In this context, a *desired state* is a particular minute in the near future where we would like to know the battery level. This becomes pivotal in the context of mobile cloud scheduling, where the scheduler should consider, among several variables, how much available energy a mobile device has to execute specific tasks. By being able to know the device's future battery level, the scheduler is able to make more intelligent decisions, and avoiding assigning a task to a device that for example will run out of battery while executing.

47.3 Evaluation

In this section, we describe the experiments performed to evaluate the model proposed in the previous section. Specifically, the estimation model is trained using the activity traces of one particular user, and then it is compared to a similar approach from the literature [14]. To assess the prediction results and compare the approaches we used the Mean Squared Error metric, which is defined as $MSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$.

The method proposed in [14] to predict the battery level is based on defining all the possible combinations of sensor states, such as energy supply connected and Wifi connected, energy supply connected and Wifi disconnected, and so on. After that, the method figures out the average time a user spends on each state, and the average battery consumption per state. Finally, the method computes the battery level by feeding that information to the following formula: $BatteryLevel = T * \sum_{i=1}^{N} p_i * B_i$ where T is the number of minutes to be predicted, p_i is the average number of minutes spent on state i and B_i is the average battery consumption per minute.

Regarding the test setup, firstly, both models are trained using the first 2 weeks of the abovementioned user's traces. Then, for each of the following days in the dataset, specifically 30 days, we pick a particular hour of the day, 12 p.m., and each model is run to estimate the battery level along the next 6 h. After that, the MSE is calculated for each curve and averaged to get the MSE per day for each model. The hour 12 p.m. is chosen because it is the time of the day on which the mobile phone has more activity, and it is when schedulers might take more advantage of the model. Besides, 6 h is a good baseline for comparing both approaches, since [14]'s model does not return good estimations in the long run as it can be seen in Fig. 47.5.

The final result of this process can be found in Table 47.4. The first two columns show the average MSE per day for both methods. It is clear that our proposed method has a lower MSE considering every day. This outcome is due to the fact that [14] is based only in the average consumption, and it does not take into account time-related factors, such as when the battery is going to be charged. Our proposed method, instead, not only takes into account previous states by learning its behavior with a Regression Model, but also predicts whether the mobile phone is going to be connected to the energy supply or not in the future. That feature helps the model figure out if the series is going up or down at a specific time. Besides, by predicting whether the screen is going to be on, the model can adapt the fastness at which the battery level decreases or increases (depending on the energy supply).

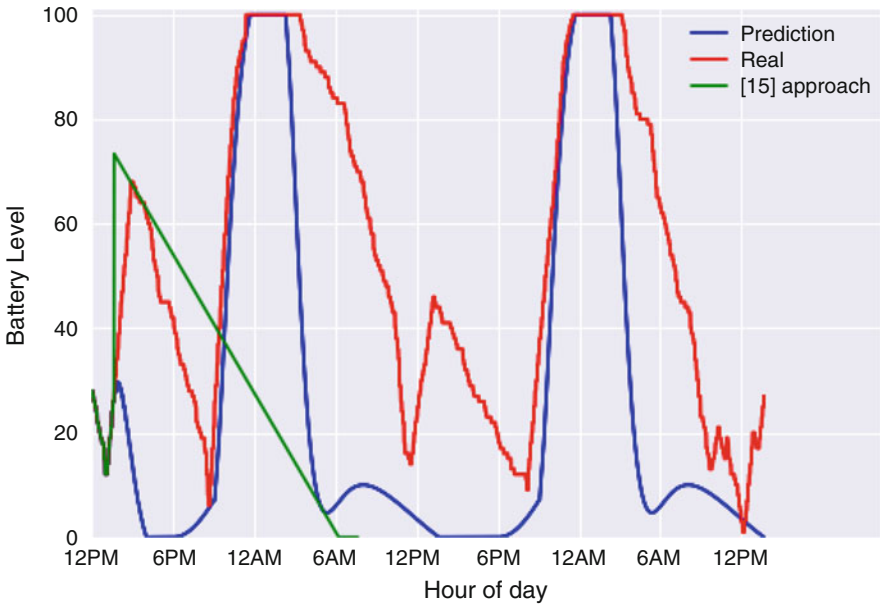

Fig. 47.5 Predicted battery level for a period of 48 h using the proposed approach and [14]

Table 47.4 Per-day average MSE of [14] and our approach

	Average MSE		Minimum MSE		Maximum MSE	
Day/approach	[14]	Ours	[14]	Ours	[14]	Ours
Sunday	33.26	18.82	9.89	1.96	55.29	44.34
Monday	35.44	21.09	24.82	6.47	50.09	49.79
Tuesday	31.99	24.44	16.93	7.20	48.69	36.58
Wednesday	38.36	26.10	18.29	4.87	55.90	47.73
Thursday	37.52	16.40	20.88	6.38	51.16	40.48
Friday	35.24	18.42	17.38	5.19	56.64	38.46
Saturday	33.59	17.58	11.42	7.28	52.53	39.68

Minimum and maximum MSE are also shown

In fact, the complementary models make our approach more precise for longer periods of times considering the test user. Figure 47.5 shows the predicted battery level for a period of 36 h using the proposed approach, i.e., from a given starting point it iteratively predicted, state by state, the final battery level after 48 h. It is clear that there is a significant visual resemblance between the real battery level and the estimated one. Moreover, the energy supply model had a very high influence in predicting the hour when the battery level is going to increase. For that particular case, [14]'s MSE is 57.67, while our approach obtained 27.86.

47.4 Conclusions

In this paper we have described a novel model to predict battery availability in mobile devices. The basic idea is to exploit past device owners activity and relevant device state variables with a two-phase approach, which includes feature extraction/selection techniques on one hand and regression models and machine learning classifiers on the other hand.

Preliminary experiments performed using mobile phone usage data from the Device Analyzer data-set and comparisons against the estimation model published in [14] yielded encouraging results. Our model was able to reduce the MSE metric, and hence estimations are more accurate. This is in line with our utmost objective, which is quantifying future energy availability in mobile devices in order to consider them as first-class resource providers in state-of-the-art edge computing paradigms such as Dew Computing.

Future work involve generalizing our results with more test cases. Concretely, we need to test the per-user accuracy of our model using device activity from several device owners. Fortunately, the Device Analyzer data-set contains many users with activity data spanning several weeks and even months. As an aside, models to fill in the potential missing activity data must be developed (e.g. days for which activity data is missing due to the device is off). In addition, other Machine Learning models can be used in order better capture the intrinsic relationships between features, so as to come up with more accurate predictions. Specifically, we will analyze a recurrent neural network (NN) known as Long Short Term Memory (LSTM) [15]. As Recurrent NN predictions depend on previous states, LSTMs are very effective for predicting state sequences [16], hence all previous states can be taken into account. Finally, we could also compare our results against battery estimation models from the industry, such as those behind battery manager applications in the Google Play Store.

Acknowledgements We acknowledge the financial support by ANPCyT through grant no. PICT-2013-0464. The first author acknowledges his MSc. scholarship in Data Science (USA) granted by Fundación Sadosky.

References

1. N. Fernando, S.W. Loke, W. Rahayu, Mobile cloud computing: a survey. Futur. Gener. Comput. Syst. **29**(1), 84–106 (2013)
2. K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems. Mob. Netw. Appl. **18**(1), 129–140 (2013)
3. M. Sharifi, S. Kafaie, O. Kashefi, A survey and taxonomy of cyber foraging of mobile devices. IEEE Commun. Surv. Tutorials **14**(4), 1232–1243 (2012)
4. S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, A. Schneider, Enabling real-time context-aware collaboration through 5G and mobile edge computing, in *12th International Conference on Information Technology-New Generations (ITNG)* (IEEE, New York, 2015), pp. 601–605
5. F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in *Proceedings of the first edition of the workshop on Mobile Cloud Computing* (ACM, New York, 2012), pp. 13–16
6. P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017)
7. M. Gusev, A dew computing solution for IoT streaming devices, in *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (IEEE, New York, 2017), pp. 387–392
8. K. Skala, D. Davidovic, E. Afgan, I. Sovic, Z. Sojat, Scalable distributed computing hierarchy: cloud, fog and dew computing. Open J. Cloud Comput. **2**(1), 16–24 (2015)
9. C. Tapparello, C.F.B. Karaoglu, H. Ba, S. Hijazi, J. Shi, A. Aquino, W. Heinzelman, Volunteer computing on mobile devices: state of the art and future, in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, pp. 153–181 (2015)
10. M. Hirsch, J.M. Rodríguez, C. Mateos, A. Zunino, A two-phase energy-aware scheduling approach for CPU-intensive jobs in mobile grids. J. Grid Comput. **15**(1), 55–80 (2017)
11. D.T. Wagner, A. Rice, A.R. Beresford, Device analyzer: large-scale mobile data collection. ACM SIGMETRICS Perform. Eval. Rev. **41**(4), 53–56 (2014)
12. I. Guyon, A. Elisseeff, An introduction to variable and feature selection. J. Mach. Learn. Res. **3** 1157–1182 (2003)
13. F.X. Diebold, G.D. Rudebusch, On the power of dickey-fuller tests against fractional alternatives. Econ. Lett. **35**(2), 155–160 (1991)
14. J.-M. Kang, S.-S. Seo, J.W.-K. Hong, Personalized battery lifetime prediction for mobile devices based on usage patterns. J. Comput. Sci. Eng. **5**(4), 338–345 (2011)
15. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
16. H. Sak, A. Senior, F. Beaufays, Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition (2014). arXiv preprint arXiv:1402.1128