IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received May 21, 2019, accepted June 15, 2019, date of publication June 27, 2019, date of current version July 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2925350

Evaluation of Machine Learning Approaches for Android Energy Bugs Detection With Revision Commits

CHENYANG ZHU[1], ZHENGWEI ZHU[1], YUNXIN XIE[2], WEI JIANG[1], AND GUILING ZHANG[1]
[1]School of Information Science and Engineering, Changzhou University, Changzhou 213164, China
[2]School of Petroleum Engineering, Changzhou University, Changzhou 213164, China

Corresponding author: Zhengwei Zhu (zhuzw@cczu.edu.cn)

This work was supported in part by the National Nature Science Foundation of China under Grant 61772090, and in part by the China Scholarship Council (CSC) under Grant CSC 201708060147.

ABSTRACT Performances of smartphones are profoundly affected by battery life. Maximizing the amount of usage of energy is essential to extend battery life. However, developers might concentrate more on the functionality of applications while ignoring the energy bugs that drain the battery during the development process. There are no quantitative approaches to detect these energy bugs introduced in this fast-paced development process. In this paper, we employ a system-call-based approach to develop a power consumption model for Android devices. Data that measure the energy consumption of mobile devices under different testing scenarios with the number of triggered system calls are utilized in the model training process. A balanced recursive feature elimination with cross-validation approach is proposed to select and rank the importance of the different system calls. Seven machine learning models are trained over the selected features with cross-validation and hyper-parameter tuning technique, where linear regression with the Lasso regularization outperforms all the other models. Then, the model is evaluated on the data set that measures the energy consumption on different revision history of the selected apps. The results show that the optimized Lasso model could detect energy bugs in the revision history of various applications. Optimization strategies are provided based on the selected features.

INDEX TERMS Energy modeling, feature selection, machine learning, code optimization.

I. INTRODUCTION

With the thriving evolution of hardware and software components of mobile devices, a lot more attention has been concentrated on promoting the performances of mobile applications while extending the battery life of smartphones. The mobile applications make use of the hardware components and built-in sensors, together with software logic, to provide useful and innovative capabilities to client users. However, the battery power supply has been an energy constraint to the smartphones regardless of the advances in battery technology and operating systems [1]. Research shows that over 18% of the Android applications in Google play market place have issues with energy efficiency, and the commercial applications also have energy consumption problems compared with the freely-available applications [2]. A lot of empirical studies has been focused on resolving the energy consumption issues of Android applications in three categories, namely building the energy consumption model, detecting energy bugs with code analysis and optimizing code structures with energy-saving practices.

However, existing approaches mainly use static code analysis and energy profiling to build energy models, detect energy bugs, and optimize applications. Developers may introduce energy bugs such as resource leak and layout defect that drain the battery during the development process. The resource leak occurs when the application does not release its required resources, and the layout defect would consume more energy to measure and draw the layout of the application [3]. There are no quantitative and efficient approaches that can detect energy bugs in the fast-paced development procedure. Moreover, developers might concentrate more on the functionality and performance of applications and ignore the energy bugs during the development process.

The associate editor coordinating the review of this manuscript and approving it for publication was Huiqing Wen.

Results in [4] show that 18% of developers would take energy consumption into account when developing software, and 14% would consider minimizing energy consumption as a requirement during development. Thus it is critical to identify the energy bugs during the development phase so that developers are aware of the changes in their source code might cause severe problems on the energy consumption of the application.

Besides, existing approaches that model power consumption with the utilization of hardware components are insufficient as they cannot interpret the power states such as tail energy. The system-call-based power modeling approach can be adopted as a more precise approach to model power consumption with the number of system calls and CPU use time of the application [5]. Using computer techniques to build models and predict/classify unseen values is becoming an essential aspect in areas such as medical image analysis, signal processing, and text categorization [6]–[8]. Based on the dataset that contains the number of system calls triggered by applications as features and the measured energy consumption as labels, our approach builds machine learning models to automate the energy measurement procedure by predicting the energy consumption in each valid commit over the revision history of the development. Firstly, a balanced feature selection algorithm is proposed to select useful features to build the energy consumption model. Then hyper-parameter tuning is used to select the optimal parameter set for each machine learning model. The performances of several machine learning models are compared with the Mean Absolute Error (MAE) and R^2 evaluation matrix. The optimized model with the best performance is then applied to the data set of [9], which collected the energy consumption data for each revision of twenty applications using the hardware mining approach in [10]. Results show that our optimized model could detect the abnormal changes in energy consumption values that identify energy bugs in the revision history, which could be used to automate the process to identify energy bugs.

This paper is constructed as follows. Section II summarizes the related work in energy modeling, energy bug detection, and code optimization in Android application development. Section III introduces the machine learning algorithms used in this paper, namely linear regression with Lasso and Ridge regularization, stochastic gradient descent regression, support vector regression, Adaboost, random forest, and gradient tree boosting. Section IV describes the framework that detects energy bugs in Android devices. Section V presents the balanced recursive feature selection algorithm. Section VI compares the performance of different machine algorithms with mean absolute error and R-squared value. The models are optimized with the hyper-parameter tuning technique. Section VII then uses the optimized model with the best performance to test the energy consumption data with revision commits. Section VIII summarizes the work and sketches some future work.

II. RELATED WORK

A. MODELING ENERGY CONSUMPTION

Hoque conducted a comparison of different energy profilers for mobile devices with the terminologies of model-based energy profiling [11]. They classify the type of power models into three different models, namely utilization-based models, instruction-based models, and system-call-based models. The utilization-based models correlate the power consumption with the utilization of hardware components. Romansky *et al.* used LSTM based time series based models to allow developers to align traces of software behavior with traces of software energy consumption [12]. Damavsevivcius *et al.* proposed a time-delay model of battery that describes the correlation between power consumption and CPU load, memory allocation, and memory release [13]. Instruction-based models apply static analysis to the code to correlate the energy consumption with instructions or APIs in the application. Hao *et al.* investigated the per-instruction energy modeling and code analysis to analyze the application usage with accuracy within 10% [14]. Hu *et al.* built the energy consumption model from method-level and API-level perspective, which guides the appropriate method and API to use to improve the energy efficiency [15]. However, some non-linear energy consumption characteristics such as tail-energy cannot be captured by the utilization-based model and instruction-based model. System-call-based models are adopted to build energy consumption models more accurately as energy consumption is correlated to the number of system calls triggered by the application. Pathak *et al.* presented a system-call-based power modeling approach that improves the accuracy of application energy estimation with the tool *eprof* [5]. To build accurate software energy consumption models, Chowdhury *et al.* developed random test generators to collect system calls triggered by the application and built regression models to predict energy consumption based on the collected data [16]. In this paper, we developed system-call based models with machine learning approaches to predict the energy consumptions of different applications over the development process.

B. ENERGY BUG DETECTION

Static analysis, as well as code analysis, have been used to detect different energy bugs during development. Pathak *et al.* investigated the characterization of no-sleep energy bug and developed tools that infer the energy bugs automatically [17]. Banerjee1 et al. categorize the energy inefficiencies into *energy hotspots* and *energy bugs* where *energy hotspots* describe scenarios where executing applications to consume a lot of battery power and *energy bugs* describe the situations where malfunctioning application prevents the smartphones from becoming idle [18]. Jiang *et al.* proposed an inter-procedure and intra-procedure analysis to detect energy bugs and layout defects automatically [3]. Damavsevivcius *et al.* presented a static analysis of energy measurement approaches by using software profiling and API based measurements [19]. Instead of identifying the energy bug in the code, our procedure identifies the commit that introduces energy bugs or hotspots in the revision history so that developers could find the buggy commit efficiently.

C. CODE OPTIMIZATION

Several approaches have been proposed to optimize apps to save energy based on the code analysis. Li *et al.* suggested to replace the light colored background areas of web applications with dark colors to reduce the power consumed by the OLED screens [20]. Li *et al.* investigated the practices such as bundling network packets and using some coding disciplines for accessing class fields and reading array length could reduce energy consumption [21]. Chowdhury *et al.* explored the HTTP/2 protocols for the web applications and concluded that HTTP/2 protocol could save energy for the networks with higher round trip time [22]. Banerjee and Roychoudhury presented a refactoring technique that can reduce the energy consumption of the open-source apps between 3% to 29% [23]. Cruz *et al.* made this re-factoring procedure automatic with static analyzing [24]. Tuysuz *et al.* developed a real-time network power consumption profiler to measure the power consumption levels of different network interfaces and proposed a handover strategy to improve energy efficiency [25]. Our work analyzes the system calls and provides code optimization strategy based on the selected features that most affect the energy consumption of an application.

III. MACHINE LEARNING MODELS

A. LINEAR REGRESSION WITH REGULARIZATION

Linear regression is one of the most used machine learning techniques to model a linear relationship between one or more variables with the corresponding result. Consider a linear regression problem with the training set (x_1, y_1), (x_2, y_2), .., (x_n, y_n) where each x_i belongs to a feature vector X with size n and each y_i is the corresponding labeled value of x_i. The linear regression fits the model $f(x, w)$ with a linear combination of the feature vector in Equation 1. The linear regression algorithm is designed to find the weights that minimize the loss function in Equation 2. In this paper, we use the Mean Square Error (MSE) as the loss function.

$$f(x, w) = w_0 + w_1 * x_1 + \ldots + w_n * x_n \tag{1}$$

$$L(y, f(x, w)) = \frac{\sum_{i=1}^{n} || y_i - X_i w_i ||_2^2}{n} = \frac{\sum_{i=1}^{n} (y_i - X_i w_i)^2}{n} \tag{2}$$

However, a complex model that achieves high accuracy on a training set would lose its predictive accuracy in the unseen data because of overfitting [26]. Lasso and ridge regression was then proposed to add penalties to the weights to balance model complexity and prediction accuracy [27]. The objective function of Lasso and Ridge is shown in Equation 3 and Equation 4 respectively. The linear regression with Lasso penalty is meant to find the weights w that minimize the loss function with the $l1$ term of weights with a constant α that multiplies the $l1$ term. The Ridge regression, on the other hand, regularize the loss function with a $l2$ term of the weights. Lasso and Ridge can be used to avoid overfitting for linear models and improve the generality of the model.

$$Lasso = argmin_w(L(y, f(x, w)) + \alpha \; || \; w \; ||_1) \tag{3}$$

$$Ridge = argmin_w(L(y, f(x, w)) + \alpha \; || \; w \; ||_2^2) \tag{4}$$

Stochastic Gradient Descent (SGD) is usually used to minimize regularized loss of linear models [28]. To minimize the loss function specified in Equation 2, the gradient descent technique is executed to update the weights in Equation 5. SGD is an iterative method to update the weights to minimize the loss function. Each iteration the weights are updated with the gradient and the step size η. SGD will converge to a global minimum if the loss function is convex. Otherwise, it converges to a local minimum [29].

$$w := w - \eta \nabla L(y, f(x, w)) \tag{5}$$

B. SUPPORT VECTOR REGRESSION

The support vector machine (SVM) is used to construct hyperplane to distinguish data from different classes [30]. Support vector regression (SVR) is proposed by Drucker *et al.* to approximate and predict the values by using a soft margin of tolerance ξ in SVM [31]. SVR is designed to find the function $f(x)$ with at most ϵ-deviation from the target y, formally presented as formula . Here ξ and ξ^* define the positive and negative soft margin allowed for the model with at most ϵ-deviation. In SVR, the goal is to minimize the function in formula 7. Here C is the penalty parameter of the error term $\sum_{i=1}^{n}(\xi_i + \xi_i^*)$.

$$\forall i \cdot y_i - w * x_i - b \leq \epsilon + \xi_i \wedge w * x_i + b - y_i \leq \epsilon + \xi_i^* \tag{6}$$

$$\frac{1}{2} \; || \; w \; ||^2 + C * \sum_{i=1}^{n} (\xi_i + \xi_i^*) \tag{7}$$

SVR uses linear hyperplane to separate the data and predict the values. However, in some cases, the distribution of data is non-linear. Thus kernel functions are used to map the non-linear separable feature space to linear separable feature space with kernel functions [32]. Radial basis function (RBF) kernel is one of the commonly used functions, which is shown in Equation 8. In the RBF kernel, $|| \; x - x' \; ||_2^2$ denotes the squared Euclidean distance between the two feature vectors x and x'. Thus RBF kernel can be used as a measurement of similarity as the value of RBF kernel decreases when the distance between two feature vectors decreases. In this paper, we evaluate the performance of linear SVR and SVR with RBF kernel functions to generate a better model.

$$K(x, x') = exp(-\frac{1}{2\sigma^2} \; || \; x - x' \; ||^2) \tag{8}$$

C. ADABOOST REGRESSION

Besides the linear regression and SVR, boosting is also one of the most powerful machine learning technique by balancing the weakness of models in each boosting iteration [33].

AdaBoost is one of the mostly used boosting algorithm, which adjusts the weights of data instances to train the regressor based on the loss function of each boosting iteration [34]. Drucker improved the AdaBoost regressor algorithm by using regression trees as weak models in the boosting steps [35]. Algorithm 1 summarizes the algorithm for the AdaBoost Regressor modified from [35]. Initially, the weight of each data is assigned to 1. During the boosting procedure, a regression machine h_m is constructed to predict x_i to $\hat{y}_i$. Then we calculate the loss for each training sample and average the losses with $\overline{L}_m$. The β that measures the confidence in the predictor is then calculated with $\overline{L}_m$. A lower value of β indicates higher confidence in the prediction. At last, when we use the boosted model for regression, we select the h_m with the lowest β_m to make the prediction. In this paper, we evaluate the effect of the number of iterations and learning rate to the performance of the energy bug detection model built with AdaBoost regression algorithm.

Algorithm 1 AdaBoost Regressors Algorithm, Modified From (Drucker, 1997)

```
1) Given:$(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in X, y_i \in Y \in IR$
   the number of iterations: M
2) Initialize the observation weights $w_i = 1, i = 1, 2, \ldots, n$
3) For $m = 1$ to $M$:
   a) Fit a regressor $h_m(x)$ to training data by minimizing the weighted error function
      $\hat{y} = h_m(x)$
   b) Compute a loss for each training sample and calculate an average loss
      $\overline{L}_m = \sum_{i=1}^{n}(\mid \hat{y}_i - y_i \mid * \frac{w_i}{\sum w_i})$
      and evaluate
      $\beta = \frac{\overline{L}_m}{1 - \overline{L}_m}$
   c) Update the data weighting coefficients
      $w_i = w_i * \beta^{[1-L_i]}$
4) Make predictions using $h_m(x)$ with lower $\beta_m$.
```

D. ENSEMBLE METHODS

Ensemble methods are learning algorithms that create a set of classifiers and take a weighted vote of the predictions of these classifiers, which consists of bagging approach and boosting approach [36]. The bagging approach builds several classifiers and averages their predictions while the boosting approach constructs the final classifier sequentially by diminishing the weakness of the classifier of each step. Random Forest (RF) is a bagging approach introduced by Breiman based on the regression tree [37], and Gradient Tree Boosting (GTB) uses the boosting approach to build a tree model sequentially [38].

Algorithm 2 shows the RF algorithm. The RF takes a random sample from the dataset and builds the regression tree based on the randomly selected features. Moreover, this process is repeated for M times, which builds M trees in the forest. Then the test data item x is fed into the model; each tree in the forest predicts a value $\hat{y}_i$. The final predicted value of $\hat{y}$ is determined in step 3), which takes the average of all the values $\hat{y}_i$ predicted by all the trees in the forest. In this paper, we use the hyper-parameter tuning approach to evaluate the impact of the number of trees in the forest on the performance of the model.

Algorithm 2 Random Forest Algorithm

```
1) Given:$(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in X, y_i \in Y \in IR$
   the number of regression trees: M
2) For $m = 1$ to $M$:
   a) Construct a CART tree $CART(x)$ with randomly selected data $x \in x_i$ with randomly selected features.
   b) Get the prediction $\hat{y}$ of each CART tree and add the CART tree to the forest $F = F \cup CART(x)$
3)
   $\hat{y} = \frac{\sum_{i=1}^{M} \hat{y}_i}{M}$
```

Algorithm 3 Gradient Tree Boosting Algorithm, Modified From (Friedman, 2002)

```
1) Given:$(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in X, y_i \in Y \in IR$
   the number of iterations $M$
2)
   $F_0(x) = argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
3) For $m = 1$ to $M$:
   a)
      $z_{im} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$
   b) Fit base learning $h_m(x)$ to $z_{im}$.
   c) Compute step magnitude multiplier
      $\gamma_m = argmin_\gamma \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma h_m(x))$
   d)
      $F_m(x) = F_{m-1}(x) + \rho \gamma_m h_m(x)$
```

Algorithm 3 shows the GTB algorithm. We want to minimize the objective function of L. The $F_0(x)$ is initialized with a gradient descent process that minimizes L with a constant

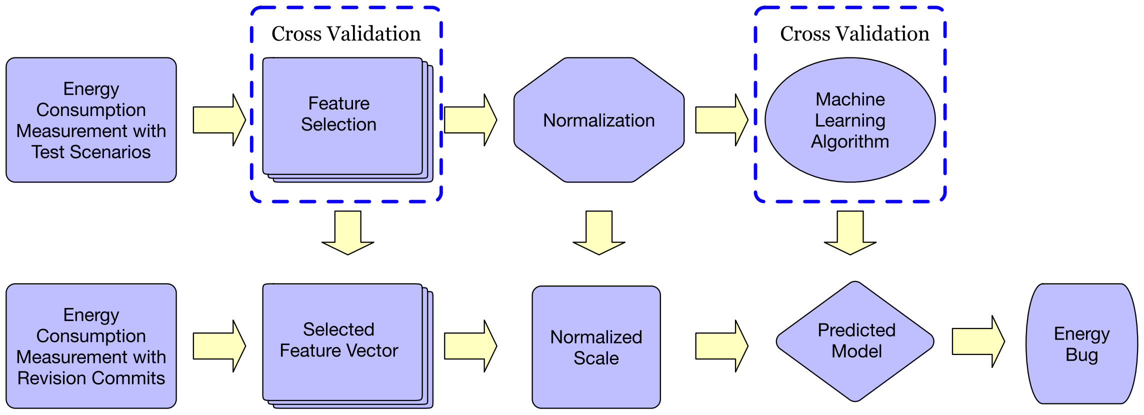

Cross Validation
Cross Validation
Energy Consumption Measurement with Test Scenarios
Feature Selection
Normalization
Machine Learning Algorithm
Energy Consumption Measurement with Revision Commits
Selected Feature Vector
Normalized Scale
Predicted Model
Energy Bug

FIGURE 1. **Revision-based energy bug detection framework.**

value of γ. We carried out M boosting iterations to train the model. Over each iteration in step 3), we first calculate the gradient z_{im} of loss function L at the point $F(x) = F_{m-1}(x)$. Then the regression tree $h_m(x)$ whose leaf nodes produce an average gradient among samples with similar features is fit to z_{im}. When the step direction is settled, the step magnitude multiplier γ_m is calculated to minimize the loss function for the samples in each leaf in step 3.c). At last, the model $F_m(x)$ is updated with the model $F_{m-1}(x)$ of the last boosting iteration and the step magnitude multiplier γ_m with a shrinkage parameter ρ that is proposed by Friedman to scale the contribution of each weak learner [39].

IV. METHODOLOGY

A. REVISION-BASED BUG DETECTION FRAMEWORK

Pathak *et al.* observed that utilization-based models could not be used to build energy consumption models in several scenarios, such as tail state, file open and closed state, socket open and closed state [5]. System-call-based models are used to overcome the limitations of utilization-based models by providing quantitative approaches to measure how the applications obtain access to the hardware components. In this paper, the numbers of system call together with the duration of CPU usage time are used as features to train power usage model that captures the precise energy consumption behavior of different applications. Machine learning is a data-based technique that learns from previous data patterns to predict or classify unseen data instances. Supervised learning that trained with the labeled training data in the form of regression and classification is an essential part of machine learning [40]. In this paper, we used the labeled energy consumption dataset with system-call based features with which we want to build the energy consumption model. Figure 1 shows the revision-based bug detection framework that detects bugs from the development revision history. First, we collect data from [16] as the energy consumption measurement from different test scenarios, which uses the number of system and GPU durations as features. In this paper, we collected 122 features to measure energy consumption, which contains the number of system calls and the number of CPU jiffies. A CPU jiffy represents the period of time that process is run by the application, which is determined by the kernel constant *HZ* [41]. We used the number of CPU jiffies to calculate the number of CPU time occupied by the application. Besides jiffies, we also consider system calls such as *socket*, *duration* to measure power usages.

After the data collection, we performed a balanced recursive feature selection with cross-validation and generated the selected feature vector. With the normalized feature vectors and the energy consumption values for each test, we compared the performance of several machine learning algorithms mentioned in Section III with cross-validation and got the model with the best performance. Then we evaluated the selected feature vector, normalized scale, and predicted model with the data collected in [9], which contains the energy consumption measurement of different applications with part of the revision commits. Abnormal rises of the measured energy consumption indicate the appearance of energy bugs in the corresponding commit. If the predicted energy consumption also reveals the abnormal rises in the same commit, then the predicted model can be used to detect energy bugs with limited manual work.

B. EVALUATION TECHNIQUE

In supervised learning, the training data might not include all the data patterns of test data. So bias and variance always exist. Sometimes overfitting would occur when the model can only predict the samples with a perfect score but fail to predict unseen examples. Cross-validation (CV) is used to test whether the induction algorithm is suitable for a given dataset. If the variance of CV estimates is approximately the same, then the induction algorithm is stable for a given dataset [42]. The CV can be applied in the feature selection phase and model hyper-parameter tuning phase. Using the k-fold CV approach, the training set is split into k sets, and the procedure is repeated for k times. Each time the model is trained with the $k-1$ folds of data and tested on the kth fold of the data. The performance of a specific feature or parameter is evaluated by averaging the measured values computed in the loop.

In this paper, we use MAE and the coefficient of determination to measure the performance of the model. Given the label set with predicted value set $(y_1, \hat{y_1}), (y_2, \hat{y_2}), .., (y_n, \hat{y_n})$, MAE is calculated as Equation 9, which could be used denote the absolute error between the predicted value and true value. The coefficient of determination is calculated as Equation 10, which could be used to indicate the proportion of the variance in the dependent variable that is predictable from the independent variable [43]. The best value of R^2 is 1. Moreover, a model with constant prediction will get an R^2 score of 0.

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n} \tag{9}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} R^2 = 1 - \frac{\sum_{i=1}^{n} (\hat{y} - \bar{y})^2}{\sum_{i=1}^{n} (y - \bar{y})^2} \tag{10}$$

V. DATA DESCRIPTION WITH FEATURE SELECTION

A. DATA DESCRIPTION

Using the energy profiling tool to measure energy usages of different applications is not precise enough as the profiling process also consumes energies. To resolve this issue, Hindle *et al.* proposed a hardware mining software repositories testbed that measures the energy consumption of mobile devices with different scenarios [10]. However, their approach requires setting up many hardware components to measure the energy consumption, including the Raspberry Pi, Android device, DC power supply, and Adafruit INA219 IC. For the Adafruit INA219 IC that measures the power usage, much manual work is required to read the value of voltage and current, then calculate the used power.

In this paper, we utilized the data collected in [16] as the training data and data collected in [9] as the test data. Chowdhury *et al.* obtained the energy consumption with the resource usage data based on the hardware-based mining software energy consumption framework. They set up Raspberry PI to run automatically generated tests on Galaxy Nexus phones and collect power usage data from INA 219. *strace* program that traces the total counts of all different system calls invoked by an application is used to generate the feature space of the training data. The measurement of power usage by INA 219 for each test of the application is used as the corresponding ground truth of energy consumption of the application. So readings of the resource usage and the corresponding energy consumption of 472 apks are used as the training data, the feature space size of which is 122. The evaluation data is collected with the same approach, which collects the energy consumption data with resource usage for 25 applications with their development commit history. The models that fit the training data is then fed to the test data. If the abnormal rises of the measured energy consumption in one commit are detected in the corresponding predicted measurement, then the model can be used to identify the commit with energy bugs introduced in the continuous development.

B. FEATURE SELECTION AND FEATURE SCALING

As mentioned in the previous section, the size of the feature space is 122, which is quite large. Training the model with this feature space will cause overfitting. Also, we want to investigate the importance of the features that affect the energy consumption value. In this paper, we proposed a balanced feature selection algorithm to generate the feature vector and rank the features with the importance that influences the energy consumption value. Algorithm 4 shows our approach that selecting the most critical features from the feature space. In our approach, CV is used to select the best feature set. The CV procedure is repeated K times, each time the data is partitioned into the training set and testing set. The model is then trained with all the features in the training set. Each feature is then ranked based on its importance. Here we define S_i with the feature subset that contains the most i important features. Then we repeat the procedure that trains the model with S_i N times and record the model accuracy. For each CV procedure, the feature subset that gets the best prediction accuracy is selected. To consider the balance and variance of the whole scheme, the CV procedure is repeated M times. The ranking of the features depends on their number of appearance in the selected features over M iterations.

Feature scaling is widely used in machine learning algorithms to avoid the situation that some features with a broad range of values dominate the objective function and make the model unable to learn from other features. In this paper,

Algorithm 4 Balanced Feature Selection Algorithm

```
1) Given training data with features of size: $N$
   the number of iterations: $M$
   the number of K-Fold: $K$

2) Initialize the dictionary: $fe$, where key is the feature and
   value is the count as 0.
3) For $m = 1$ to $M$:
   a) For $k = 1$ to $K$:
      i) Partition data into training set as $K \setminus k$ and
         testing set $k$ randomly.

      ii) Fit the model to all the features and rank the
          features based on their importance.

      iii) For each subset size $S_i, i = 1, \ldots, N$
           A) Train the model using features in $S_i$

           B) Determine model's accuracy
   b) Calculate the accuracy of cross-validation score
      and get the best $S_i$.
   c) Construct the feature list $f$ with the number spec-
      ified by $S_i$.
   d) Add $f$ to the dictionary $fe$ with:
         $\forall i \cdot i \in f \Rightarrow fe[i] += 1$
4) Rank the features in $fe$ based on the count of each
   feature.
5) The feature list is the features in $fe$ whose count is not 0.
```

TABLE 1. Feature descriptions.

Feature	Description	Feature	Description
duration	time spend on auto-generated tests	gettimeofday	get the data and time
tot_intrpt	total number of software interrupts	dup	duplicate an open file descriptor
nice	number of CPU jiffies for niced processes	getuid32	get user identity
user	number of CPU jiffies for normal processes	major_fault	number of major page faults
system	issue a command	connect	connect a socket
madvise	give advice about use of memory	cntx	number of context switches
shed_yield	empty the processor	recvfrom	receive a message from a socket
sendto	send a message to a socket	bind	bind a name to a socket

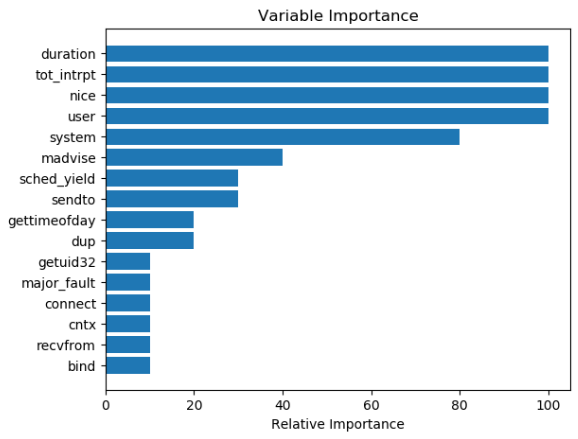


FIGURE 2. Feature importance with lasso regularization based on algorithm 4.

we used the z-score normalizing that removes the mean of the data and scales the data to the unit variance. Given the feature set F, we use $f \in F$ to denote one feature within the feature set. Given the training data with a size of n, we transform the values of each feature as Equation 11.

Based on Algorithm 4, we performed feature selection on the training data with the linear regression with Lasso regularization. Figure 2 shows the result of feature importance with the ranking of the features. Table 1 provides a detailed explanation of the selected features to model energy consumption. Figure 2 shows that the length of the test case influences energy consumption mostly, which makes sense. The longer the time the tests take, the more energy the application will consume. Results also show that the number of software interrupts and context switches affect energy consumption. When the processor switches the work on one task to another task, the state of volatile data like program counter, registers and memory need to be saved, which needs the energy to perform the task. Also, if the code change invokes more CPU jiffies, then more energy will be consumed. Developers can examine this small set of features for possible optimization of the code structures.

$$\bar{f_i} = \frac{\sum_{j=0}^{n-1} f_{ij}}{n}$$

$$s(f_i) = \sqrt{\frac{\sum_{j=0}^{n-1}(f_j - \bar{f_i})^2}{n}} \quad \forall j \cdot j \in f_i \Rightarrow f_{ij} = \frac{(f_{ij} - \bar{f_j})}{s(f_i)} \quad (11)$$

VI. COMPARE MACHINE LEARNING ALGORITHMS FOR ENERGY MODELS

A. TUNING DIFFERENT MACHINE LEARNING MODELS

As mentioned in Section III, machine learning models might have one or more parameters for different objective functions. It is essential for supervised models to select appropriate parameters to achieve high prediction accuracy and generality. Hyper-parameter tuning is one of the approaches that find the tuple of hyper-parameters that generates the optimal model that minimizes the objective function [44]. In this paper, we use hyper-parameter tuning together with the 5-fold CV to get the best hyper-parameter set of each model and compare the prediction ability of different machine learning models. Table 2 provides the hyper-parameter that requires tuning for each model. During the hyper-parameter tuning, grid search is used as the approach to search through a specified subset of hyper-parameter space. However, the grid search approach is an exhaustive searching algorithm which takes a lot of time and computation to find the optimal parameter set. Thus we first evaluated the search range of parameters by using R^2 as the cross-validation score.

Figure 3 shows the process that determines the search range of penalty α in linear and SGD regression with Lasso and Ridge regularization. Results show that R^2 of Lasso regularization is decreasing when the value of penalty α increase for the linear regression. A larger value of α will cause the linear model to lose its generality. So we restrict the search range of α for Lasso regularization within 0-0.02. Also, R^2 of Ridge regularization stay the same for the linear regression. So we define the search range for α for Ridge regularization within 0-0.1. As SGD regression updates the weights over each iteration, so the R^2 has small disturbance with the changes of α penalty. Overall R^2 stays the same for the Lasso regularization, but it decreases with the increase of the α penalty for the Ridge regularization. Thus we determine the search range for SGD regression over Lasso regularization as 0-0.1 and Ridge regularization as 0-0.02.

Figure 4 shows the process that determines the search range of penalty C, kernel coefficient and tolerance for support vector regression. Results show that in SVR with RBF kernel, R^2 increases with the increase of penalty C within 0-40 and the value converges to a stable value. Thus we restrict the search range for C in RBF kernel as 50-100. Moreover, it is evident that R^2 comes to peak value within the range 0-0.1 of kernel coefficient. So we determine the range for kernel

TABLE 2. Hyper-parameters in different machine learning models with search range, optimal setting and cross-validation score and standard deviation.

Machine Learning Model	Tuned Parameters	Search Range	Parameter Setting	CV score+/- std error
Linear Regression with Lasso	alpha	0-0.2	0.01	0.952+/-0.006
Linear Regression with Ridge	alpha	0-0.1	0.05	0.951+/-0.007
SGD Regression with Lasso	alpha	0-0.1	0.01	0.951+/-0.006
SGD Regression with Ridge	alpha	0-0.02	0.01	0.951+/-0.006
SVR with RBF Kernel	Penalty C	50-100	90	0.9+/-0.01
	Kernel coefficient	0-0.1	0.08	0.58+/-0.01
SVR with Linear Kernel	Penalty C	10-20	10	0.951+/-0.005
	Tolerance	0.0001-0.01	0.01	0.949+/-0.005
AdaBoost	Number of iterations	80-100	90	0.88+/-0.02
	Learning rate	0-2	0.5	0.88+/-0.02
Random Forest	Number of Trees	70-100	100	0.892+/-0.02
Gradient Tree Boosting	Learning Rate	0-0.4	0.2	0.892+/-0.02

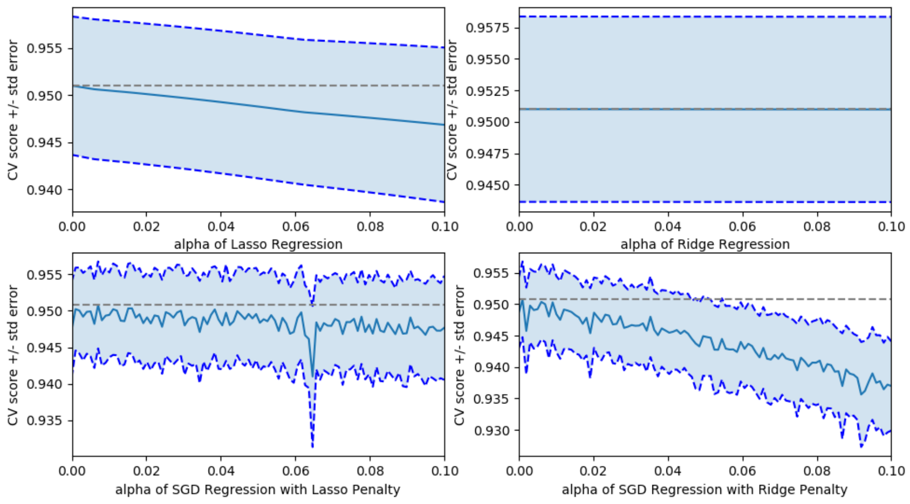


FIGURE 3. Hyper-parameter search range for linear and SGD regression with lasso and ridge regularization.

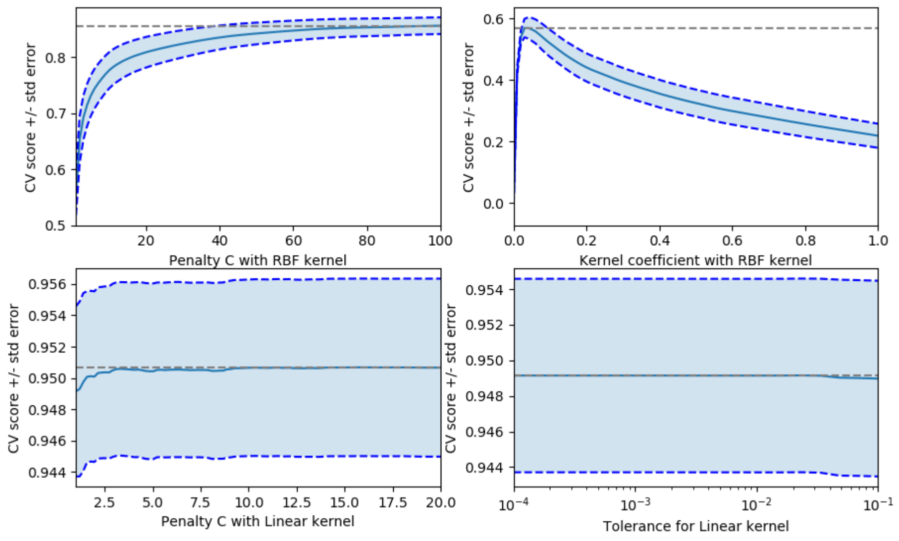


FIGURE 4. Hyper-parameter search range for support vector regression with RBF and linear kernel.

coefficient as 0-0.1. As to the SVR with linear kernel, R^2 has small disturbance for the penalty C and tolerance. So we determine the search range for the penalty as 10-20, and tolerance as $10^{-4} - 10^{-2}$.

Figure 5 shows the process that determines the search range for AdaBoost, Random Forest, and Gradient Tree Boosting. First, we evaluate the search range for learning rate and the number of iterations for AdaBoost Regression. Results show

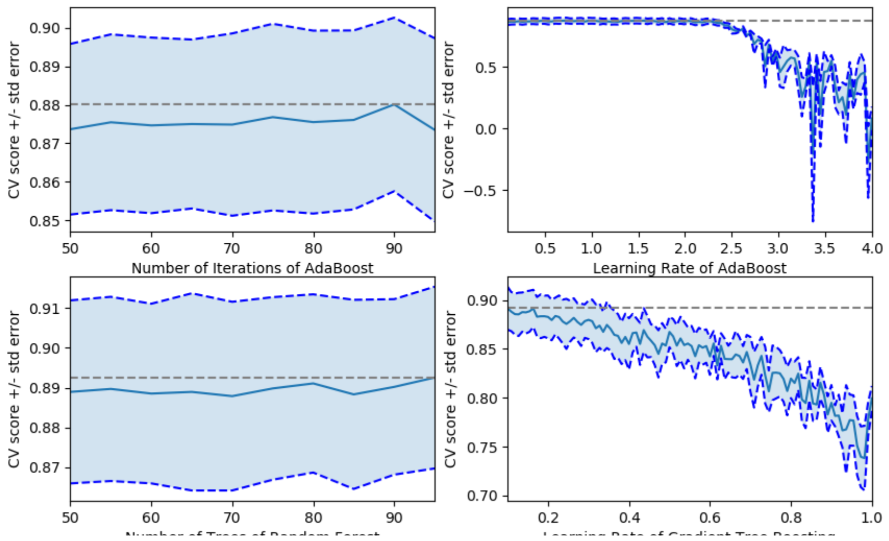


FIGURE 5. Hyper-parameter search range for AdaBoost, random forest and gradient tree boosting.

that there is a small disturbance of R^2 for the number of iterations, but R^2 decreases sharply with the increase of learning rate. It is shown that a large value of learning rate will cause the model to overfit. It is better to increase the number of iterations with a small value of learning rate. So we restrict the search range for learning rate as 0-2 and number of iterations as 80-100. The number of trees is an essential parameter that affects the performance of random forest. Results show that there is no apparent variance for the R^2 value with the increase of the number of trees. However, too many trees in the random forest will cause overfitting. So we restrict the number of trees as 70-100. During our experiment, the result shows that the variance of R^2 of GTB is small with the change of the number of iterations. However, R^2 will decrease evidently when the learning rate is within 0.6-0.8. As a result, we restrict the learning rate of GTB as 0-0.4.

B. ENERGY MODEL PERFORMANCE WITH DIFFERENT MACHINE LEARNING ALGORITHMS

Table 2 provides the CV score as well as the standard deviation error for each machine learning algorithm with the optimal parameter set. The result shows that linear regression with Lasso regularization has the best performance compared with all the other algorithms. Compared with Ridge regularization, Lasso regularization can improve the model generality. Also, linear regression works better than SGD regression. As SGD updates its weights continuously, the performance is not stable.

Moreover, overfitting could occur if the number of iterations is too large. The SVR model with linear kernel performs better than the one with the RBF kernel, which indicates that the energy model is linear to the selected features. The regression-tree based models have less generality compared with linear models as their R^2 is around 0.88-0.89, while the R^2 of the linear model is around 0.95-0.96. So the linear models have better generality compared with regression tree-based models.

TABLE 3. Comparison of different machine learning models.

Machine Learning Model	MAE	R-square
Linear Regression with Lasso	**9.35**	**0.94**
Linear Regression with Ridge	9.36	0.93
SGD Regression with Lasso	10.24	0.94
SGD Regression with Lasso	10.52	0.93
SVR with Linear Kernel	9.48	0.90
AdaBoost Regression	11.12	0.86
Random Forest	10.85	0.87
Gradient Tree Boosting	10.46	0.88

R^2 only describes the generality of a model. We use MAE to evaluate the prediction error of each model. As the SVR with RBF kernel has poor performance based on the previous experiment, we no longer assess it in the investigation. In the analysis that compares of performance of different machine learning algorithms, we split the training data into a training set and a test set randomly. As there are only 472 readings of data, so we took 80% as the training set and 20% as the test set. The analysis is replicated ten times with different random seed to balance the effect of random seeds in the model. The training set and test set is preprocessed with the selected features and specified feature scaler. Each model is trained with the optimal parameter set and the training set. Then the trained model is tested in the test set. We calculated the MAE and R^2 between the predicted energy and measured energy over each iteration. Next, we average the results in Table 3. Results show that linear regression with Lasso has the smallest MAE and highest R^2 compared with all the other machine learning algorithms, which means that linear regression has the least prediction error and has the best generality. The MAE of linear regression is around 9.35, which is about 5%-10% of error as the range of measured energy is around 100-200. Also, the coefficient of determination value

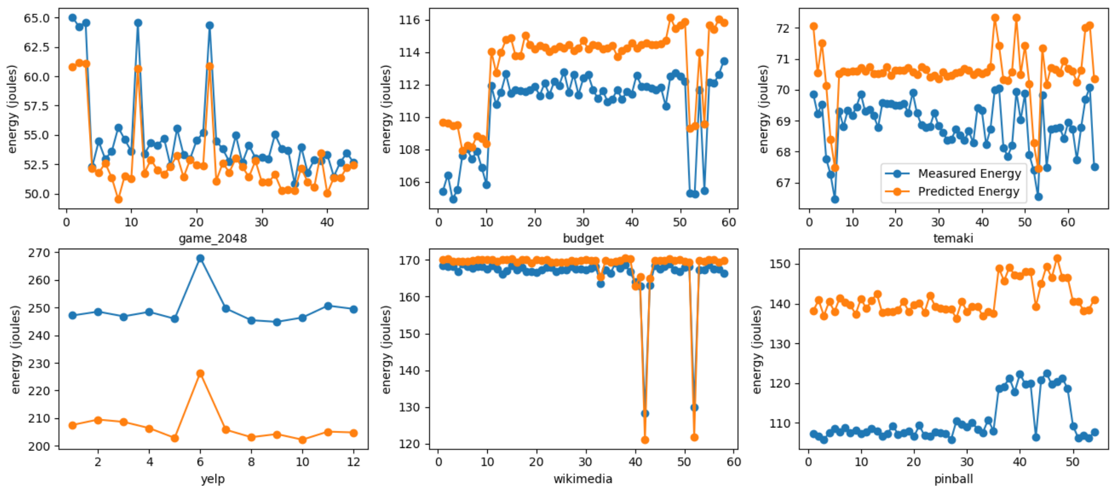


FIGURE 6. Selected applications with predicted energy and measured energy.

of 0.94 shows the model has good generality. The result of other algorithms shows either higher MAE or lower R^2, which indicates that linear regression with Lasso is the best option to build the energy model.

VII. MODEL EVALUATION AND OPTIMIZATION STRATEGY

A. MODEL EVALUATION BASED ON REVISION HISTORY

Based on results shown in Section VI, we use linear regression with Lasso as the algorithm to train the energy model. After we trained the model with the training data, we evaluate the model with the data in [9]. This dataset contains the resource usage and energy measurement for twenty application over different revision history. Figure 6 shows the selected applications with energy measurement, which have abnormal rises during revision history. For example, the 11th and 22nd commit of *game_2048*, the 6th commit of *yelp*, 35th commit of *pinball*, have exceptional rises of energy consumption compared with the other commits. Some energy bugs might be introduced into the application during these commits. Then we applied our model on this data set with the selected feature and specified feature scaler, the unusual rises and falls can be detected by our model successfully. Besides, our model can also mimic the energy consumption pattern over the revision history. Take *temaki* application as an example. *Temaki* is an Android application for creating simple lists quickly. The figure shows that there are several sharp rises and falls for the measured energy consumption. Our model can capture these patterns successfully. With the help with our model, developers no longer require power meters to measure voltage and current to calculate power consumptions for their applications. They only need to write automatically generated tests and collect the resource usage data for each commit. Then they can use the model to detect where there are energy bugs introduced in the latest commit compared with the measured energy data in previous commits.

B. OPTIMIZATION STRATEGY BASED ON SELECTED FEATURES

Besides the energy model that captures the energy consumption pattern over revision history, the selected features can also guide developers to optimize their code with less energy consumption. Based on Figure 4, *nice* and *user* are two important features that affect the energy consumption. The GPU jiffy stands for the time assigned for a process without any interrupts. The number of CPU jiffies for the niced and normal process indicates the CPU usage by the processes. Chowdhury *et al.* also mitigate the complexity in energy modeling with different power consuming states by counting the number of GPU jiffies. Some techniques can be used by developers to reduce the usage of CPU, namely reducing the sampling frequency of sensors, putting some complicated calculations on servers instead of on the client side, setting GPS as coarse localization. The wake locks should always be released once they are acquired to reduce the CPU usage time. In addition, features such as *sendto*, *recvfrom*, *connect*, *bind* also affect the energy consumption. These features are related to sockets that communicate to the Internet with TCP or UDP. To reduce the number of sockets, optimization for the network related functions could be carried out. For example, the partially-downloaded data could be cached to avoid repeatedly redownloading the data. The data calls with similar goals can be batched into one data call to reduce the number of sockets. Also, developers can reduce the system call to get data and time. Code optimization can be done to reduce the number of major page faults.

VIII. CONCLUSIONS AND FUTURE WORK

Continuously energy modeling has been a challenging task for Android developers. Massive manually measurement and calculation is required for the previous energy measurement work for Android applications. To ease the step that measures energy consumption with power meters, we built the energy consumption model with the resource usage data generated from automatically generated tests. In this paper, we brought up a balanced feature selection algorithm that selects the most critical features for Android energy consumption. Our algorithm selects 16 out of 122 features in the system-call-based feature space and ranks the features with their importance to the final model. Optimization strategies for Android applications are provided based on the selected features. Different machine learning algorithms are compared based on the MAE and R^2 evaluation matrix. Each machine learning model is trained with the optimal parameter set generated from the hyper-parameter tuning approach. The CV results show that the linear models perform better than non-linear models and regression tree based models as the linear models can achieve the R^2 score with over 0.95 while the non-linear models and regression tree based models can only achieve 0.88.

To summarize, linear regression with Lasso regularization has the best performance compared with Ridge regularization, support vector regression, regression-tree based models. Results show that the model trained with the linear regression with Lasso regularization on the test set has the MAE as 9.35 and R^2 as 0.94, which is the best score among all the models. We also use this model to capture the pattern of measured energy consumption over revision history. Figures show that the model can detect abnormal energy changes effectively, which then could be used to replace the energy measurement step with power meters.

Our paper uses resource usage of 472 applications as the training data to train the model. More data could be collected to better train and test the model with cross-validations. A Test-as-a-Service (TaaS) can be created based on the current Lasso model. The front-end allows developers to upload Android apks to test the energy consumption over each commit. The back-end server can use Raspberry PI to run tests and collect resource usage data from the submitted apks and predicted the energy consumption value with the model proposed in this paper. Moreover, the collected data can also be used as part of the training data and increase the generality of the model. Also, our model only captures the abnormal rises of energy consumption, which identifies the commit that introduces the energy bug. However, the specific code that introduces the energy bug cannot be identified with the model. More work can be done to analyze the call graph of Android applications with static analysis and incorporate the code analysis into our framework and guide the developers to resolve energy bugs more efficiently. Furthermore, the current tests that collect resource usage do not include localization tests. Some future work can be done to model energy consumption with GPS or cellular localization.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers who provided helpful suggestions to improve the quality of the paper.

REFERENCES

[1] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An empirical study of the energy consumption of Android applications," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Oct. 2014, pp. 121–130.

[2] C. Wilke, S. Richly, S. Götz, C. Piechnick, and U. Aßmann, "Energy consumption and efficiency in mobile applications: A user feedback study," in *Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber, Phys. Social Comput.*, Aug. 2013, pp. 134–141.

[3] H. Jiang, H. Yang, S. Qin, Z. Su, J. Zhang, and J. Yan, "Detecting energy bugs in Android Apps using static analysis," in *Proc. Int. Conf. Formal Eng. Methods*, Oct. 2017, pp. 192–208.

[4] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Softw.*, vol. 33, no. 3, pp. 83–89, May 2016.

[5] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. 6th Conf. Comput. Syst.*, Apr. 2011, pp. 153–168. doi: 10.1145/1966445.1966460.

[6] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[7] Y. Sun, P. Babu, and D. Palomar, "Majorization-minimization algorithms in signal processing, communications, and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 3, pp. 794–816, Feb. 2016.

[8] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, 2002.

[9] S. A. Chowdhury and A. Hindle, "GreenOracle: Estimating software energy consumption with energy measurement corpora," in *Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR)*, May 2016, pp. 49–60.

[10] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "GreenMiner: A hardware based mining software repositories software energy consumption framework," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, May 2014, pp. 12–21.

[11] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Comput. Surv.*, vol. 48, no. 3, p. 39, Feb. 2016.

[12] S. Romansky, N. C. Borle, S. Chowdhury, A. Hindle, and R. Greiner, "Deep green: Modelling time-series of software energy consumption," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 273–283.

[13] R. Damasevicius, J. Toldinas, and G. Grigaravicius, "Modelling battery behaviour using chipset energy benchmarking," *Elektronika Ir Elektrotechnika*, vol. 19, no. 6, pp. 117–120, 2013.

[14] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 92–101.

[15] Y. Hu, J. Yan, D. Yan, Q. Lu, and J. Yan, "Lightweight energy consumption analysis and prediction for Android applications," *Sci. Comput. Program.*, vol. 162, pp. 132–147, Sep. 2018.

[16] S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, "GreenScaler: Training software energy models with automatic test generation," *Empirical Softw. Eng.*, vol. 20, pp. 1–44, Jul. 2018.

[17] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services*, Jun. 2012, pp. 267–280. doi: 10.1145/2307636.2307661.

[18] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and Hotspots in mobile apps," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2014, pp. 588–598.

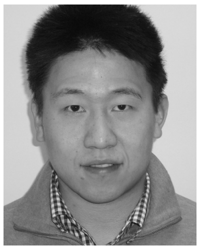
[19] R. Damaševicius, V. Štuikys, and J. Toldinas, "Methods for measurement of energy consumption in mobile devices," *Metrol. Meas. Syst.*, vol. 20, no. 3, pp. 419–430, Sep. 2013.

[20] D. Li, A. H. Tran, and W. G. J. Halfond, "Making Web applications more energy efficient for OLED smartphones," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 527–538.

[21] D. Li and W. G. J. Halfond, "An investigation into energy-saving programming practices for Android smartphone app development," in *Proc. 3rd Int. Workshop Green Sustain. Softw.*, Jun. 2014, pp. 46–53.

[22] S. A. Chowdhury, V. Sapra, and A. Hindle, "Client-side energy efficiency of HTTP/2 for Web and mobile app developers," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng.*, Mar. 2016, pp. 529–540.

[23] A. Banerjee and A. Roychoudhury, "Automated re-factoring of Android apps to enhance energy-efficiency," in *Proc. IEEE/ACM Int. Conf. Mobile Softw. Eng. Syst.*, May 2016, pp. 139–150.
[24] L. Cruz, R. Abreu, and J.-N. Rouvignac, "Leafactor: Improving energy efficiency of Android apps via automatic refactoring," in *Proc. IEEE/ACM 4th Int. Conf. Mobile Softw. Eng. Syst.*, May 2017, pp. 205–206.
[25] M. F. Tuysuz, M. Ucan, and R. Trestian, "A real-time power monitoring and energy-efficient network/interface selection tool for Android smartphones," *J. Netw. Comput. Appl.*, vol. 127, pp. 107–121, Feb. 2019.
[26] C. Schaffer, "Overfitting avoidance as bias," *Mach. Learn.*, vol. 10, no. 2, pp. 153–178, Feb. 1993.
[27] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc., B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
[28] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, Sep. 1951.
[29] K. C. Kiwiel, "Convergence and efficiency of subgradient methods for quasiconvex minimization," *Math. Program.*, vol. 90, no. 1, pp. 1–25, 2001.
[30] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[31] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Proc. Adv. Neural Inf. Process. Syst.*, Jun. 1997, pp. 155–161.
[32] 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," *J. Mach. Learn. Res.*, vol. 11, pp. 1471–1490, Jan. 2010.
[33] R. E. Schapire, "The boosting approach to machine learning: An overview," *Nonlinear Estimation Classification*, vol. 171, pp. 149–171, May 2003.
[34] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997
[35] H. Drucker, "Improving regressors using boosting techniques," in *Proc. ICML*, vol. 97. 1997, pp. 107–115.
[36] T. G. Dietterich, "Ensemble methods in machine learning," in *Proc. Int. Workshop Multiple Classifier Syst.*, Dec. 2000, pp. 1–15.
[37] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[38] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.
[39] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, pp. 1189–1232, Oct. 2001.
[40] C. E. Rasmussen, "Gaussian processes in machine learning," in *Proc. Summer School Mach. Learn.*, Feb. 2003, pp. 63–71.
[41] *Time*. Accessed: Feb. 15, 2019. [Online]. Available: http://man7.org/linux/man-pages/man7/time.7.html
[42] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, Aug. 1995, pp. 1137–1143.
[43] R. G. Carpenter, "Principles and procedures of statistics, with special reference to the biological sciences," *Eugenics Rev.*, vol. 52, no. 3, p. 172, Oct. 1960.
[44] M. Claesen and B. D. Moor, "Hyperparameter search in machine learning," Feb. 2015, *arXiv:1502.02127*. [Online]. Available: https://arxiv.org/abs/1502.02127



CHENYANG ZHU was born in Changzhou, Jiangsu, China, in 1990. He received the B.S. degree from the Huazhong University of Science and Technology, in 2012, the M.S. degree in computer science and information from the University of Pennsylvania, Philadelphia, PA, USA, in 2014, and the Ph.D. degree in electronics and computer science from the University of Southampton, Southampton, U.K.

From 2013 to 2014, he was a Research Assistant with the xLab, University of Pennsylvania. Since 2014, he has been a Software Engineer with OSIsoft LLC. He is currently with the School of Information Science and Engineering, Changzhou University. His research interests include ensemble methods of machine learning, formal methods, integrated formal methods, data visualization, and time modeling.

ZHENGWEI ZHU was born in Changzhou, Jiangsu, China, in 1963. He received the B.S. degree from Southeast University, Nanjing, China, in 1980, the M.S. degree from the South China University of Technology, Shanghai, China, in 2014, and the Ph.D. degree from the Nanjing University of Science and Technology, Nanjing.

From 1999 to 2007, he was an Assistant Professor with the School of Information Science and Engineering, Changzhou University. Since 2008, he has been a Full Professor with the School of Information Science and Engineering, Changzhou University. He has more than 50 publications, more than ten of which are indexed by SCI and EI, and holds ten patents. His research interests include computer networks, intelligent measurement, and control systems. He managed and completed more than ten provincial, municipal, and enterprise projects, one of which is funded by the National Nature Science Foundation of China.

YUNXIN XIE was born in Jingzhou, Hubei, China. She received the B.S. and M.S. degrees in petroleum science from Yangtze University and the Ph.D. degree in mineral prospecting and exploration from the Chengdu University of Technology, in 2018. She is currently a Lecturer with the School of Petroleum Engineering, Changzhou University.

WEI JIANG was born in Huaian, Jiangsu, China. He received the B.S. degree from the Changzhou University Huaide College, Changzhou, China. He is currently pursuing the M.E. degree with the School of Information Science and Engineering, Changzhou University. His research interests include indoor position, pedestrian navigation, and smartphone applications.

GUILING ZHANG was born in Xinyang, Henan, China, in 1993. She received the B.S. degree from the Changzhou University Huaide College, in 2017. She is currently pursuing the M.S. degree with the School of Information Science and Engineering, Changzhou University. Her research interests include static analysis and Android application testing.