MLStar: Machine Learning in Energy Profile Estimation of Android Apps

Benjamin Gaska
University of Arizona
bengaska@cs.arizona.edu

Chris Gniady
University of Arizona
gniady@cs.arizona.edu

Mihai Surdeanu
University of Arizona
msurdeanu@cs.arizona.edu

ABSTRACT

Improving the energy efficiency of smartphones is critical for increasing the utility that they provide to the users. With most mobile operating systems, users are responsible for managing their phone's battery efficiency by utilizing the various settings provided by the operating system, as well as selecting energy-efficient apps. However, current app marketplaces do not provide users with information about app energy efficiency, which makes it challenging for the user to make informed decision when selecting an app. This paper presents a novel machine learning approach to estimate app energy efficiency by utilizing textual information available in the Google Play store such as an app's description, user reviews, as well as system permissions. Our detailed analysis of the resulting system shows that hardware permissions, app description, and user reviews correlate well with energy efficiency ratings. We evaluate five models that represent popular classes of machine learning algorithms in their ability to predict energy efficiency ratings. Finally, we compare our approach to gold truth ratings obtained by the actual energy profiling of the app, demonstrating that the proposed system is able to estimate an app's energy efficiency within less than 1 point on the 1 – 5 scale provided by the profiler, without requiring any kind of profiling.

CCS CONCEPTS

- **Human-centered computing** → *Ubiquitous and mobile devices*;
- **Hardware** → *Power estimation and optimization*;

KEYWORDS

energy, mobile, machine learning

ACM Reference Format:

Benjamin Gaska, Chris Gniady, and Mihai Surdeanu. 2018. MLStar: Machine Learning in Energy Profile Estimation of Android Apps. In *EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '18)*, November 5–7, 2018, New York, NY, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3286978.3287011>

1 INTRODUCTION

Smartphones have become a ubiquitous part of modern society and a vast number of apps allows the users to accomplish almost all daily computation and communication task. We use apps to access the Internet, watch videos, play games, read documents,

and accomplish many other tasks on the go. More and more apps populate our phones and demand higher processing performance as the users are running more apps with more features. This in turn has a negative impact on a battery life. Since advances in battery technology continue to lag behind the demands placed on computing performance, energy efficiency has become a critical consideration in the design of mobile systems. In addition, user awareness of energy efficiency is increasing and users are looking for alternatives to prolong their smartphone operation.

Energy management techniques aim to optimize the overall operation of the system based on the running apps. While system optimization can improve overall system energy efficiency, energy bugs or inefficiencies in the apps can dominate energy usage and significantly degrade the battery life. Subsequently, the developers are becoming more and more energy conscious and are utilizing numerous energy profilers [9, 19, 29] to optimize energy efficiency of their app. While most developers are trying to design energy efficient apps, many apps still suffer from numerous inefficiency due to bugs, lack of understanding of energy efficiency by developers, or too extensive number of features and tasks that may not be necessary. Subsequently, the users are left in charge for selecting energy efficient apps for their smartphone.

The app markets present users with app features, permissions, and usability rating, but do not have any metric for energy efficiency, which prevents users from making energy conscious decisions of what app to download and use from numerous alternatives that offer similar features. The EStar profiler [9] attempts to alleviate the problem of missing energy efficiency ratings by profiling and utilizing crowdsourcing of user profiling information to present energy efficiency of many apps in the market. Crowdsourcing accelerates coverage of apps and allows the users to select energy efficient apps and contribute to ratings of apps that are not yet rated. However, the EStar marketplace currently covers only approximately 2000 apps, less than 1% of the 2.8 million apps in the Play store.

Clearly, a more scalable approach is required to give the users some hint of the app's energy efficiency. While an app's description or permissions can hint about energy efficiency, a regular user may not be able to derive the energy efficiency expectation based on those factors. To address this challenge, we propose, develop, and evaluate a scalable Machine Learning (ML) system (*MLStar*) that estimates an app's energy efficiency using only the textual and meta information publicly available on Google Play store. MLStar uses information such as user reviews, the app's description, and Android system permissions to derive and present energy efficiency rating to the users. Subsequently, we make the following contributions in this paper: (1) We show that the energy efficiency of an app can be accurately estimated using solely publicly-available textual data and Android user permissions; (2) We identify and characterize app features that correlate with energy efficiency; and finally (3) Develop a system to automatically estimate efficiency of the app

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MobiQuitous '18, November 5–7, 2018, New York, NY, USA

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-6093-7/18/11...\$15.00

<https://doi.org/10.1145/3286978.3287011>

Figure 1: App growth in Google Play store.

that is within less than 1 point of EStar rating, on average, without any profiling.

2 MOTIVATION

Increase in computing power of smartphones have led to explosion in software apps to provide advanced communication and computation tasks. As of September 2016 the Google Play store had over 2.8 million apps [3] and Figure 1 shows the rapid growth in apps since the store's opening. In 2016 alone there have been on average over 50,000 new apps added each month. This high growth rate makes it impossible for users to make energy efficient selections from a large number of apps that perform the task needed by the user. Furthermore, the users are using more and more apps to accomplish their daily communication and computation tasks. As of 2014 study, users had on average 95 apps installed [1], and this number is likely to keep increasing.

Increase in app computing needs and communication needs is increasing the pressure on mobile device's battery life and the manufacturers are including larger and larger battery capacities in their phones. Figure 2 shows the increasing battery capacity trends in the Samsung Galaxy S line of smartphones. The improvements in battery energy density are slow and the increase in battery capacity is gained at the expense of the battery size and resulting phone weight increase. In addition, the battery capacity in Galaxy S line doubled between Galaxy S and the newest Galaxy S7, while average battery life has stayed flat. As a result, it is critical to improve energy efficiency of apps and provide the users with energy rating of apps, so the user can make a conscious decision about their needs and the battery life of their devices.

EStar [9] allows the user to monitor energy efficiency of apps installed on their smartphones. EStar keeps track of the foreground and background execution with respect to computation, communication, and screen energy usage. The profiles of the apps are also crowdsourced, allowing EStar to provide overlay on top of Play store that shows energy efficiency rating of the listed apps. The app energy rating is shown on the range from 1-to-5, 1 being the least energy efficient. This service allows the users to make a more energy conscious decisions when selecting among comparable apps. For an app to receive a rating, it has to be profiled first by at least one of the users that is currently using EStar. Profiling can be challenging due to the large number of apps and the potentially much smaller EStar user base as compared to users in Play store.

Figure 2: Battery capacity in Samsung Galaxy S line.

Figure 3: Training and estimation in MLStar.

Subsequently, EStar covers approximately 2,000 apps versus 2.8 million apps in Google Play store.

These challenges led us to explore a different approach for energy efficiency estimation, using machine learning (ML) to exploit correlations between information available on the Play store and energy efficiency. In particular, we are exploring natural language features taken from user reviews, developer descriptions of each app, as well as permissions demanded by the app. This ML based system provides us with a scalable approach that can be rapidly deployed to cover all existing apps in Play store as well as new apps. In addition, our system can provide information about which app features, based on description or permissions, can be problematic for energy efficiency further expanding the understanding of energy optimization challenges.

3 MLSTAR ARCHITECTURE

They key observation in this paper is that textual information that is available on Google Play store can be correlated to the app's energy efficiency. Figure 3 presents the architecture of the MLStar system that exploits the correlation to estimate energy efficiency rating of the apps in Play store. The system has two major components, a model-training component and the energy-estimating component of a previously unseen app, which can be further divided into five operations:

- **EStar Extractor** provides the energy ratings for the rated apps that will serve as ground truth during the training of the ML model. It also provides a list of the rated apps to the Play store crawler.

- **Play Store Crawler** fetches publicly available information from Play store, such as permissions, reviews, and description for each app rated by EStar.
- **Feature Extractor** transforms the data from Play store into discrete features suitable for the training of an ML classifier. The same extractor is used during training and estimation.
- **Machine Learning** generates an ML model that correlates features to the energy rating obtained from EStar.
- **Energy Estimator** uses the generated model to estimate a previously unseen app's energy rating based on the extracted features.

3.1 Extracting Information

We rely on two sources of information for our models: EStar energy rating and information from Play store. We extract a list of apps and corresponding energy ratings (1 to 5 stars) from EStar. The Play store crawler uses the list of apps from EStar to extract corresponding textual information about each app from Play store. We collected all available fields from Play store such as permissions, app description, user reviews, user ratings, app URL, developer URL, price, etc. We discarded unique identifiers to each app such as app URL, the developer URL, etc. We also discarded price and user ratings due to lack of variation. For example, every app from our dataset that is provided by EStar is free. In both cases the data does not allow correlations between different apps.

Table 1 shows the information and examples the MLStar system uses for training and estimation. App Description is the text provided by the developer to describe their app. This text contains words and phrases that can provide information to the machine learning models about the functionality of the app. Similarities in words and phrases across data points allow us to associate otherwise unrelated apps which aids in estimating energy efficiency. Similarly, user reviews can provide additional app feature description not mentioned by the developer, or focus on dominating app features highlighting the app usage scenarios. User reviews can provide criticism indicating high resource usage or short battery life. Finally, the sheer number of reviews provides much broader view, as compared to single description by the developer.

Any app running on Android must provide a set of system permissions to the user indicating what system components the app will utilize. The permissions can be highly indicative of the application behavior and resulting energy efficiency. For example, network access combined with camera and multimedia permissions may indicate high network usage to upload the pictures or video that can significantly degrade energy efficiency. Subsequently, this most direct description of the potential app behavior is the final key component of the information from Play store considered by MLStar.

3.2 Feature Extraction

The first step in machine learning is feature extraction where raw data is converted to a n -dimensional numerical vector known as a feature vector. These feature vectors encode the information from our raw data in a numerical format suitable for learning. For example, one dimension of this vector indicates how many times the word "camera" appeared in the app's description. Another dimension may be a Boolean (0 or 1) value indicating if the app has (or does not have) permission to read text messages. These vectors

from each field are then concatenated into one feature vector for each app.

Table 2 lists the features used for training and estimation by MLStar. We extract the textual features as unigrams, words in the text counted one by one, and bigrams, counting text in two-word long contiguous segments. All textual data is encoded as a "term frequency/inverse document frequency" (*tf-idf*) vector, which is a normalized vector to promote terms frequently occurring in the given app while, at the same time, demoting terms that are common across all apps, such as "the" or "of". Permissions are the Android system permissions requested by each app, and are encoded as a Boolean vector that notes whether the permission is requested, or not.

Textual features, presented in first four rows in Table 2, used the same feature extraction algorithm. First, the textual data is tokenized into individual words. Next, low information words, known as stop words, are removed from the text. These consist of function words and other very common words such as determiners (e.g., "the"), pronouns (e.g., "I"), prepositions ("of"), etc. These words supply little information at best and can be a source of noise during training at worst, and so they are commonly removed entirely. Finally, punctuation and other such non-word indicators are removed from the data and unigrams and bigrams are generated. Unigrams model the situation where word sequence does not matter, i.e., the "bag of words" model [17], but only occurrence frequency of the word is important. For example, in a user review for an app that said, *"I love this game I keep playing it over and over and over"*, each word would be counted separately, resulting in counts of 3 for "over" and 1 for "game".

Bigrams provide a simple way to model word sequences by capturing only immediate associations. The order of words is captured by the bigram as it encodes the sequential aspect of natural language. For example, the bigram "always online" appearing in an app description tells more about the energy efficiency of the corresponding app than the two unigrams "always" and "online" appearing in the description without information how they relate to one another. Further, bigrams provide a simple way to model negation, e.g., "not efficient" is very different from "not" and "efficient" independently. We count bigram occurrence using a sliding window of two words. We tested higher order n -grams but they did not lead to increased accuracy, likely due to the sparsity of the textual data.

All textual features (unigrams and bigrams) are added to the feature vectors with a value computed using the term frequency – inverse document frequency (*tf-idf*) formula:

$$tf-idf(w, t) = tf(w, t) \times \log\left(\frac{1}{df(w)}\right)$$

Where w is unigram or bigram to be added and t is the corresponding text for which we compute the feature vector (review of description). tf represents frequency of w in the text t (e.g., 3 for the word "over" in the example above) and df is the document frequency of w across the collection of texts, i.e., how many documents contain w overall [17]. Intuitively, the *tf-idf* promotes terms that are frequent in the current text, while, at the same time demoting common terms that are frequent across documents (due to the $\frac{1}{df(w)}$ component). For instance, many of the apps are games, and their description contains the word "gameplay", which is assigned a low weight accordingly. Finally, for non-textual permission data, we use a list of all 235 existing permissions compiled by the Pew

Description	Example
App Name	Star Wars Force Collection
EStar Rating	2.5
App Description	...Create your own battle formations. Each card has its own combination of card skill, ...
User Reviews	It's a nice game but I don't like the part where the trial period ends.
Permissions	Read Messages Send Messages, Access Contacts, Network Access, GPS Location

Table 1: Information about the app used by MLStar.

Feature	Encoding	Example
Description - Unigrams	<i>tf-idf</i> vector	"the best game ever" is transformed into a vector "the", "best", "game", and "ever" with weights computed using the <i>tf-idf</i> formula
Description - Bigrams	<i>tf-idf</i> vector	"the best game ever", becomes three bigrams, "the best", "best game", and "game ever"
Reviews - Unigrams	<i>tf-idf</i> vector	"I love this app", becomes "I", "love", "this", and "app"
Reviews - Bigrams	<i>tf-idf</i> vector	"I love this app", becomes "I love", "love this", and "this app"
Permissions	Boolean vector	[0,1,1,0,0,1,1,0,0,1].

Table 2: Features and encoding used by MLStar.

Research Center [25] and encode them as a 235-dimension Boolean vector for each app. This process is repeated for the data associated with each app in our dataset, producing a distinct feature vector for each app.

As a concrete example, Figure 4 shows the data for a single dimension in the feature vector. The "Record Audio" permission is the most informative of all features and in the graph we can see a small tendency for those apps that have the permission to have a lower rating. Over the entire data set these little correlations among the different dimensions allow for models that can accurately predict energy consumption of each app.

3.3 Model Training

Note that the task of predicting energy ratings could be modeled in two distinct ways: (a) we could model the rating to be predicted as a discrete variable, e.g., low/average/high energy consumption, or (b) we can model it as a continuous variable, e.g., a real value between 1 and 5. When the value to be predicted is discrete, we call the problem a *classification* problem (the former setup); when it is continuous, we call it a *regression* problem (the latter case). In this work, we choose the more general setup, and model the task as a regression problem.

The architecture introduced in Figure 3 allows any regression algorithm to be applied on the feature vector extracted from a given app. In general, regression models learn a function h that operates over a feature vector f , and is a predictor of the variable to be estimated (energy rating in our case). Again, there are two classes of models: *linear* models, which approximate the function h with a linear polynomial over f , and *non-linear* models, which, as the name implies, are capable of learning more complex, non-linear functions. Both approaches learn their corresponding function h to minimize prediction errors over the examples given in the training dataset. Non-linear models have the capability of addressing more complex tasks, where the interactions between features are more complicated. But they also have the disadvantage of "hallucinating" models [10], i.e., learning functions that closely replicate the training dataset but fail to generalize to new data points. This typically happens on small and/or noisy datasets, where the models assign high importance to features (or feature interactions) that appear

Figure 4: EStar ratings example.

relevant in the training data, but are not important in the wild. This process is called *overfitting* [10]. On the other hand, linear models do not have same representational power because they are limited to simpler, linear functions. But this limitation comes with an advantage: they over fit less, because they cannot replicate all the intricacies seen in training. In this work we explore both classes of models (see Methodology).

Learning these functions consists of two major pieces: a loss function and an optimization algorithm. The loss function takes the prediction made by the model on the training data. It compares this to the label supplied with the training data and generates a loss value. One of the most common of these is the quadratic loss function:

$$F(p) = w(t - p)^2$$

Where p is the prediction made by the model, t is the target label, and w is a weighting parameter. This also shows a common feature in loss functions in that the loss grows exponentially as the prediction difference increases. With this loss function defined the goal becomes to minimize this function across the training data.

A common optimization algorithm for minimizing this value is called Stochastic Gradient Descent (SGD). In SGD each update is based on a sample or single element from the entire training set. This allows us to scale the algorithm to larger data sets. The idea of this is that to minimize the loss we can take steps down the gradient of the function by updating the weights in the mapping function.

Figure 5: Example of Stochastic Gradient Descent. Each iteration goes down the gradient of the subspace until it converges at a minimum [26]

On each iteration of gradient descent, the loss function is calculated and the derivative of this loss function is calculated to determine the current sign of the gradient. The larger the value of the loss function the larger the step made in the weight update. Intuitively, predictions that are further from correct lead to greater correction than close predictions. This is done until moving the weights in any direction would lead to an increase in the loss function. At that point we are at a minima. Figure 5 shows an example of SGD in a 3-dimensional space. This final function is then used to make predictions about new data.

Once a model has been trained, i.e., a predictor function h has been learned, we can use it to predict the energy rating of any new app from the Google Play store. During the energy estimation phase (right block in Figure 3), we crawl new apps from the Play store, and extract their features using the same components used during training. The resulting feature vector is fed into the predictor function h to yield a predicted energy rating. To evaluate the accuracy of the proposed approach, we compare these predicted ratings against gold truth ratings from EStar on a set of apps that were held-out during training.

4 METHODOLOGY

Data collection consisted of mining application list with energy ratings from EStar and corresponding information from Google Play for each app in the list, as shown in Figure 3. To obtain a balanced dataset, we sampled across all app categories. The amount of apps rated by EStar is considerably lower than apps available in Google Play, so certain categories had few apps available, sometimes as few as one app per category. Ultimately, we downloaded 1,400 apps across 24 and divided this dataset into a training set of size 1,200 and a testing set of size 200. That is, the later 200 apps were held out during training, and were only used to estimate the prediction accuracy of the resulting models on unseen data. Table 3 shows an extended data breakdown for relevant fields from our data collection.

To gather data from the Google Play store, we created a web crawler that automatically collects information from each app's page. We designed the crawler to function in two ways: (1) direct targeting of apps fed to the web crawler as the URL of their store page, and (2) by a breadth-first walk through the store gathering information for as many apps as possible. In this paper, we use the first method to collect fields for specific app list from EStar. Apps can have hundreds of thousands of reviews; therefore, we limited the crawler to collect only the first 20 pages of user reviews

generated from the app store page. This limit was chosen to prevent excessive run times, while still gathering a broad range of reviews about an app.

We implemented the machine learning classifier using the scikit-learn library [20], which is a widely-used library for machine learning in the Python ecosystem. It provides a variety of tools for data preprocessing and feature extraction, and implements most common machine learning classifiers. For the experiments in this paper, we report results using the following common five regression models:

- *Support Vector Machines (SVM)* map the training examples (e.g., apps) as points in a n -dimensional space (in the most common situation n is the dimension of the feature vectors introduced before) such that the different labels (e.g., energy ratings in our case) are maximally separated from each other. We trained two different SVM models:
 - *Linear SVM* requires that the function separating the different labels (ratings) be linear. That is, it learns to estimate energy ratings using a first-degree polynomial function h that linearly combines the values in the feature vector f : $h = \sum_{\text{all features } i} f_i \times w_i$, where f_i is the value of the feature i in the feature vector, and w_i is the weight assigned to this feature, as learned during training.
 - *Non-linear SVM* using a Radial Basis Function (RBF) kernel, where the separating function is non-linear, following the RBF kernel function.
- *Adaptive Boosting (AdaBoost)* sequentially trains a large amount of decision trees, where each decision tree is trained to improve upon the mistakes made by the previous one. Each decision tree is a complete classifier, capable of predicting an energy rating given an app's feature vector. To achieve a single prediction, the predictions of all decision trees are combined using a weighted voting mechanism, where the weights are driven by the performance of each decision tree during training.
- *Random Forest* builds a collection of decision trees similarly to AdaBoost. Unlike AdaBoost, these decision trees are independent of each other; each one is trained on a sample extracted from training, using a subsample of features for the decisions in each node.
- *Linear Regression* is a classic ML algorithm, which fits a line to the training data that minimizes the prediction error on the training samples. The function learned is similar to the one learned by the linear SVM above, but is learned using the Stochastic Gradient Descent algorithm discussed previously.

Note that two of these models (Linear Regression and linear SVM) are linear (thus, as discussed in Section 3, have less representational power but are less prone to overfitting). The other three are non-linear models, which have opposite properties. We also implemented a baseline that assigns to each app in the held-out dataset the mean energy rating from the apps in the training dataset. For each of the above models we used the default learning parameters from their scikit-learn implementation (i.e., we performed no tuning of hyper parameters). Each model was trained using the same preprocessing and the same feature vectors. The models were then used to estimate energy ratings for the 200 held-out apps.

For evaluation purposes, the estimated energy ratings were compared to the rating given to that app by EStar. The evaluation measure was Mean Squared Error (MSE) across all 200 predictions

Category	Number In		Permission Count		Average # of Reviews	Estar Score	
	Training	Testing	Average	Std. Dev.		Average	Std. Dev.
Games	316	47	10.33	4.58	108.84	3.63	0.94
Communication	146	20	23.50	12.29	108.18	2.89	1.41
Tools	101	15	19.16	17.68	103.29	2.45	1.43
Social	73	11	19.41	9.72	104.22	3.94	1.37
Entertainment	70	15	10.37	4.38	112.94	3.56	1.24
Education	62	8	8.25	4.14	113.57	3.52	1.20
Productivity	48	10	17.68	10.54	95.77	2.89	1.51
Transportation	48	6	13.98	6.58	109.77	2.83	1.36
News/Magazines	39	6	13.48	4.86	97.20	3.74	0.89
Travel/Local	36	3	16.46	7.57	100.92	3.66	0.98
Lifestyle	34	12	14.76	7.16	105.52	3.53	1.34
Music/Audio	31	8	12.64	5.57	111.07	3.66	1.05
Photography	31	7	13.55	6.44	107.05	3.43	1.22
Weather	25	3	13.00	6.64	98.14	2.0	1.14
Health/Fitness	25	2	13.96	5.90	106.66	3.27	1.14
Media/Video	23	6	13.55	5.08	107.31	3.29	1.23
Shopping	23	6	15.48	5.55	108.20	3.63	1.14
Personalization	22	2	22.66	14.87	109.45	3.75	0.88
Finance	21	4	12.56	4.65	104.88	4.25	1.13
Books/Reference	11	3	10.00	114.0	114.0	4.07	1.23
Medical	6	2	11.37	5.82	110.25	3.25	0.93
Business	4	2	13.33	6.92	93.0	2.83	1.54
Maps/Navigation	2	1	25.33	0.47	76.0	3.0	0.40
Comics	1	1	3.00	1.0	63.0	2.25	0.25
Libraries/Demo	1	0	9.00	114.0	114.0	1.0	0.0
Dating	1	0	15.0	0.0	54.0	5.0	0.0
All Categories	1200	200	14.63	10.01	106.30	3.37	1.12

Table 3: Summary of apps in the collected dataset.

(thus, a lower value is better). For example, assume a held-out set of two apps, ranked by EStar with 3 and 4 stars. If our model's predictions are 4 and 3.5 respectively, then the overall MSE is:

$$\frac{(4 - 3)^2 + (3.5 - 4)^2}{2} = 0.625$$

5 RESULTS

Using the methodology and features described above we trained all the above models and tested their accuracy on the 200 held-out apps. Figure 6 shows the MSE of the models' predictions on this dataset. The figure shows that the baseline model, which assigns to each app in testing the average rating observed in training has a MSE of 1.130. Two out of the three non-linear models (AdaBoost and the SVM with RBF kernel) do not outperform this baseline considerably, which suggests overfitting. The non-Linear SVM had a MSE of 1.107; AdaBoost a MSE of 1.112. As discussed in Section 3, overfitting occurs when the models fail to generalize by fitting the training data too closely. This is common for non-linear models when the training data is small or noisy. These results indicate that we are in this situation with our current dataset. This is not surprising: the Google Play store is very large, and our dataset represents a very small sample of all possible apps. These models ended up assigning high importance to features that were arbitrarily relevant in our training dataset but their relevance did not generalize beyond these examples.

Regression Models
Figure 6: Comparison of different training models.

However, both linear models (Linear Regression and linear SVM), as well the simpler non-linear model (Random Forest) outperform the baseline considerably. Linear Regression had a MSE of 0.940, and the linear SVM performed the best out of all models, with a MSE of 0.927. This is a 22% reduction error (relative) over the baseline predictor. The Random Forest algorithm performed slightly

Figure 7: Error distribution of model predictions.

worse, but still better than the other non-linear models, with a MSE of 0.987. This result strongly supports our hypothesis that we can indeed estimate an app’s energy rating from using solely its publicly-available information. Further, this suggests that the relation between features extracted from public information and energy rating can be modeled using simple, linear functions. This is encouraging, as these functions can be more directly interpreted by human end-users (see Section 5.2 for a further discussion).

Lastly, to understand if these differences are indeed statistically significant (i.e., they are likely to hold on other datasets or we were “just lucky” on the current dataset), we implemented statistical significance analysis of the results using non-parametric bootstrap resampling over 100,000 samples of the testing set. This test entails sampling over the testing set of apps for 100,000 rounds. During each round, 200 testing apps were chosen using random sampling with replacement. The MSE is determined for those apps (for the given model to be analyzed), and compared to the MSE for the baseline predictor. If our observation (i.e., that our model outperforms the baseline) holds in more than 99% of the rounds, we can claim statistical significance with a p -value < 0.01 (intuitively, the p -value is the percentage of new datasets where our observation fails) [11]. The p -values for the Random Forest, SVM with a linear kernel, and for the Linear Regression were determined to be ≤ 0.01 indicating that our results are indeed statistically significant for these models.

5.1 Characterizing Prediction Error

Figure 7 shows a histogram of the prediction error (i.e., how much off in any direction is the model from the ground truth) for the testing set. This was computed using the best performing classifier, the linear SVM. A negative error here means that the model predicted an energy rating worse than the EStar rating; a positive error means that it predicted a better rating.

As expected given its relatively good performance, most errors for the linear SVM model cluster around 0, meaning that the model is generally correct. With respect to higher error values, the figure indicates that the model has a small negative bias: more errors (55%) fall to the left of 0. This indicates that the model puts more importance on features that impact negatively the battery life, rather than on features that indicate the opposite. We detail this observation in the next subsection.

5.2 Ablation Analysis

To understand which individual feature groups are most important for the prediction of an app’s energy rating we implemented an

Figure 8: Ablation analysis comparing the MSE of the system with the linear SVM model (“All Features In”) against the system when one feature group is removed.

ablation analysis experiment. In ablation experiments, individual feature groups are independently removed from the model and the system performance is measured under the new configuration. The lower the performance of the latter system, the more important the removed feature is.¹

Figure 8 summarizes the results of this experiment by showing the top 10 feature groups with the most impact on overall performance. The figure shows that all but one of the top 10 predictive features are Android permissions. As introduced in Section 3, there are 235 possible permissions, each of which must be individually requested by an app. All permissions fall into one of two classes: hardware permissions and user data permissions.

Four of the permissions in the figure fall under user data permissions: “read your text messages”, “read sensitive log data”, “precise location”, and “record audio”. These permissions are about accessing background data and being able to read or to write to specific parts of memory. The first two permissions indicate that the app is likely to read text messages and sensitive log data, which further indicates that the app is doing background scanning of information on your system. This process can significantly impact energy usage. The latter two permissions require interfacing with data from hardware on the phone, the GPS and WiFi for “precise location”, and the microphone for “record audio”. Similarly, this causes large energy drain over time, especially in cases where the phone is being periodically awoken by the app to check this information.

The remaining five permissions fall under hardware. These are permissions that allow the app to interact with hardware or modify system level information. Three of the five hardware permissions

¹Note that for models that generate linear functions such as Linear Regression or linear SVM, there is a more direct strategy to understand feature importance: simply inspect the linear function learned and identify the features assigned the larger weights (or smaller, depending on what is measured). The advantage of this approach is that it does not require training and evaluating a multitude of different models. The disadvantage is that it only holds for linear models. To ensure generality, in this work we focus on ablation analyses, which are independent of the type of the underlying ML model.

Figure 9: Improvements in prediction vs. training size.

(“send wap-push-received messages”, “body sensors”, “read subscribed feeds”) involve accessing the wireless networking features of the system. It is not surprising that these permissions are relevant predictors of energy consumption: wireless communication is the single most energy intensive process on mobile devices. “Control vibrations” is similar in that it allows for access to the motor controlling phone vibration, yet another source of energy consumption. The outlier in this set is “modify secure system settings”. This permission allows an app to access low level system settings that are not normally modified by apps. This is not normally requested by most apps in the store. Further analysis shows that among the 12 apps that request this permission, they request on average 46.5 permissions, when the average number of permissions requested by an app in our dataset is 14.63. In this case there seems to be a correlation between this permission appearing and the app’s overreaching in its requests, which, further, is an indicator of being energy hungry.

Finally, the description unigram feature group is the *tf-idf* vector that encodes all words that appear in user descriptions. Natural language text is a rich source for inference in machine learning models such as this. For example, the occurrence of certain words like “graphics” or “online” serves as strong (indirect) indicator of the possible energy usage of that app, by indicating certain energy-hungry functionality.

5.3 Learning Curve

Figure 9 shows training behavior with respect to the size of the training dataset for all proposed models as number of training apps increases by 100 apps at a time. Such a curve allows us to identify trends in the data and models such as how rapidly the prediction accuracy is increasing, and whether (and when) learning has leveled off. Once again there is a significant gap between the worst non-linear models (AdaBoost and SVM with RBF kernels) versus the other models. AdaBoost shows extreme fluctuations with every new data chunk added to its training set. This supports the idea that the AdaBoost classifier is overfitting to the training data. The non-linear SVM has a different problem, in that its performance is nearly constant as new training data is introduced. There is a small increase in performance over the full set, but it does not add up

to any significant improvements over time. This suggests that the non-linear SVM constantly overfits: the model fits some (apparent) optimum over the training dataset, but this does not generalize to the testing dataset.

The other three models perform more in line with expectations. The linear SVM and Linear Regression models perform very similarly to one another at all steps, both learning better as new data is introduced. As the amount of training data increases the linear SVM begins to distinguish itself, widening the gap between it and its nearest competitor. The Random Forest also follows a similar learning curve as to the two linear models, but performing worse as the amount of training data increases. The similarity of the three curves indicates that, indeed, there are common correlations in the data that all three models are learning. However, each model has differing sensitivities to the noise in the data, and different ways of learning, which manifest in their different prediction capabilities over the testing dataset.

All in all, this analysis suggests that the current size of the training dataset is not sufficient. Figure 9 shows that as more data is added most models continue to improve their performance. This means that more training data is required to overcome the high variance between different apps. The Google Play store consists of over 2.8 million apps, and the dataset we were able to collect contains only 1,400 apps. We expect that with a larger data set it becomes possible to even more accurately model the large space of apps available.

5.4 Energy Rating by Category

For further understanding of MLStar’s behavior, we plot in Figure 10 its average prediction by category, compared against the gold truth, i.e., the EStar average rating. Our dataset contains apps from 26 different categories from the store. However, eight of these categories were represented by two or fewer apps in our testing dataset, and were excluded from the analysis in Figure 10. The figure shows that there is a wide difference (approximately 2.5 points) between the categories at the extremes of the chart, the Weather category and the Finance category, the worst and best rated categories respectively. The finance category includes apps such as Geico Mobile and Bank of America Mobile. In general, these are apps that tend to be a means to access data only when the users request it. They tend to be passive, performing little background computation when the user is not directly interacting with them. Thus, their energy requirements tend to be minimal. On the other hand, apps in the Weather category, e.g., the The Weather Channel Mobile and MyRadar Weather Radar, feature a large amount of background network traffic as they constantly are updating for user location and current weather conditions. Such apps that frequently awaken the phone to perform network communications are a known problem in the mobile energy community, and are well known for their negative impact on battery life.

The comparison between MLStar’s predictions and EStar’s rating indicates that the most extreme difference between MLStar and EStar is in the Books category. As discussed previously, our model has a tendency to underpredict energy efficiency, and this is what we observe here again: our model predicts a full point below on average from the EStar rating for this category. Our analysis indicates that the divergence comes from the types of permissions

Figure 10: The average energy rating comparison of EStar and MLStar.

requested by apps in this category. There is a high level of commonality between the much lower energy efficient Media and Video category and Books. Both commonly request permissions such as Prevent Device From Sleeping and many WiFi related permissions. Their usage of these permissions differs greatly between categories, with less background computation being necessary for the Books category than Media and Video. However, there are considerably more apps in Media and Video than Books, about twice as many in our dataset, which drives the model's predictions more towards the ratings of the less efficient Media and Video category.

In general, we see consistent predictions across categories. The language and permissions used in any given category tends to be distinct enough to cluster similar apps together for energy prediction. This is consistent with the previous work of Banz et al. [22], which also showed that permission information allows for accurate prediction of app's category. It is important to note that adding category information as an explicit feature does *not* impact performance. Our explanation is that a relatively small number of categories (corresponding to a small number of apps) contain potential signal, and it is drowned by noise. Only five of the 24 categories have, on average, performance more than 0.5 points from the average. Three of these five categories (Communication, Tools, Social) make up 26% of the total data points, but these three categories are all internally highly variable in their energy rating: the standard deviation of energy scores in these categories is in the top five out of all categories. Tools for instance includes both a calculator app, which has high energy efficiency, and a anti-virus monitoring system, which does frequent scans of the entire file system for malware. These two facts mean that category information is a muddled signal that provides limited information to the ML models. On the other hand, our approach lets the ML system build *de facto* clusters of apps, grouped by features that are consistently relevant for energy rating.

6 RELATED WORK

Optimizations of energy efficiency of apps that are running on smartphones start with the developer. Developers can use numerous energy profilers to optimize the app for energy efficiency before it is deployed to the users. Energy profiling techniques fall into two categories, software and hardware-assisted profilers. A hardware-assisted profilers [12, 13, 16, 27] rely on external equipment to

monitor energy consumption during application profiling. Clearly, those profilers are not easily available to the end user. Software profilers, on the other hand, estimate energy by building a model based on system traces [6, 13, 19, 29] and they can be used much more easily the end user. Recently mobile system manufacturers are including energy monitoring devices into their system design blurring the line between the software and hardware-assisted profilers, and releasing their own profilers to assist developers in performance and energy optimizations [12, 13, 16, 27].

The energy optimizations are moving into the realm of monitoring and optimizing the application behavior in the user systems. Crowdsourcing where the profilers on the user systems send exchange data are gaining popularity as they allow accelerating application characterization and offering instantaneous benefit to the users for existing apps [8, 9, 18]. Crowdsourcing has been also applied to optimize system setting to automatically improve energy efficiency [21] for end users. Manufactures are also following the trend by releasing their own profilers that utilize energy monitoring hardware to offer analysis of the application energy efficiency to the end user [2]. Subsequently, the users are more and more aware of energy efficiency and can take the steps to improve efficiency of their smartphone.

MLStar is using machine learning to accelerate ranking of the application energy efficiency without requiring extensive profiling. Machine learning has been most widely utilized to detect malicious apps by finding trends in APIs and library calls in the code to identify unusual behavior [4, 5, 23]. Run-time frameworks [7, 24] execute alongside other applications on the device and collect behavioral information, this information is used to learn a model that identifies unexpected behavior in real time. All of these techniques rely on low-level information about the app, and require either full analysis of the code or data collection during runtime. Alternatively, natural language processing has been applied at much higher level by analyzing application description, using Latent Dirichlet Allocation – a popular machine-learning algorithm for learning the topics of texts, to determine the described behavior that the app is telling the user it performs [14]. The description then has been compared to static analysis of the app to check if the actual behavior of the app matches the claimed behavior.

Machine learning has also been applied to sentiment analysis, using feature from natural language to describe the positive and negative feelings that users have towards a subject [15]. Machine learning techniques have been applied to extract specific topic users are discussion about an app, and showing whether sentiment is positive or negative. MLStar takes this information and applies it to uncovering new information rather than a summarization task. Finally, machine learning has been applied to characterize relation between apps usage and the battery discharge to predict smartphone operating time [28].

7 CONCLUSIONS

In this paper, we have introduced MLStar, a machine learning technique that provides an estimation of any app's energy efficiency using only information that is publicly available in Google Play store. This is considerably different from previous energy profiling techniques, which require tedious profiling to offer energy estimates. We believe we are the first to propose a machine learning solution that does not require app profiling for this important task. Our simpler approach guarantees scalability to the very large number of apps available in mobile app stores. Our broad-range of features include natural language information extracted from the app description and user reviews, as well as features extracted from the app permissions. Combining all these features yields a model that is capable of predicting an app's energy rating within less than one point from the gold truth on the EStar scale of 1 to 5, a 22% relative improvement over a baseline that assigns to each new app the average rating seen in training.

Our detailed analysis highlights several important observations. For this task, linear regression models outperform non-linear models, suggesting that the latter class of models are prone to overfitting on this data. Further, we highlight features that are important for prediction. We show that both non-textual features including both hardware permissions (e.g., access to sensors) and user data permissions (e.g., recording audio), as well as textual features extracted from the app description are important for accurate predictions. We also show that energy efficiency does not meaningfully correlate with a given app's popularity among users. This indicates that the users are not making the most energy conscious decisions, as it is simply not available, and energy rating can go a long way to educate the users about energy efficiency and motivate the developers to further focus on efficient application design. Finally, this work demonstrates that not only it is possible to estimate an app's energy efficiency rating ahead of time, but, also, that it is possible to explain to the end user why the machine learning produced a certain estimation (e.g., apps that require permission to record audio tend to be energy hungry).

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1551057.

REFERENCES

- [1] Android users have an average of 95 apps installed on their phones, according to yahoo aviate data. <http://thenextweb.com/apps/2014/08/26/android-users-average-95-apps-installed-phones-according-yahoo-aviate-data/>. Accessed: 2017-01-14.
- [2] App tune-up kit. <https://developer.qualcomm.com/software/app-tune-up-kit>. Accessed: 2017-01-14.
- [3] Number of available android applications - appbrain. <http://www.appbrain.com/stats/number-of-android-apps>. Accessed: 2017-01-11.
- [4] AAFER, Y., DU, W., AND YIN, H. Droidapiminer: Mining api-level features for robust malware detection in android. In *International Conference on Security and Privacy in Communication Systems* (2013), Springer, pp. 86–103.
- [5] ARP, D., SPREITZENBARTH, M., HUBNER, M., GASCON, H., AND RIECK, K. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS* (2014).
- [6] BANERJEE, A., CHONG, L. K., CHATTOPADHYAY, S., AND ROYCHOUDHURY, A. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2014), ACM, pp. 588–598.
- [7] BURGUEIRA, I., ZURUTUZA, U., AND NADJM-TEHRANI, S. Crowddroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices* (2011), ACM, pp. 15–26.
- [8] CHANDRA, R., FATEMIEH, O., MOINZADEH, P., THEKKATH, C. A., AND XIE, Y. End-to-end energy management of mobile devices.
- [9] CHEN, X., JINDAL, A., DING, N., HU, Y. C., GUPTA, M., AND VANNITHAMBY, R. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), ACM, pp. 40–52.
- [10] DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM* 55, 10 (2012), 78–87.
- [11] EFRON, B., AND TIBSHIRANI, R. J. *An introduction to the bootstrap*. CRC press, 1994.
- [12] FENSEL, A., TOMIC, S., KUMAR, V., STEFANOVIC, M., ALESHIN, S. V., AND NOVIKOV, D. O. Sesame-s: Semantic smart home system for energy efficiency. *Informatik-Spektrum* 36, 1 (2013), 46–57.
- [13] FLINN, J., AND SATYANARAYANAN, M. Powerscope: A tool for profiling the energy usage of mobile applications. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on* (1999), IEEE, pp. 2–10.
- [14] GORLA, A., TAVECCHIA, I., GROSS, F., AND ZELLER, A. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (2014), ACM, pp. 1025–1035.
- [15] GUZMAN, E., AND MAALEJ, W. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)* (2014), IEEE, pp. 153–162.
- [16] JUNG, W., KANG, C., YOON, C., KIM, D., AND CHA, H. Devscope: a nonintrusive and online power analysis tool for smartphone hardware components. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (2012), ACM, pp. 353–362.
- [17] MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [18] OU, Z., DONG, J., DONG, S., WU, J., YLÄ-JÄÄSKI, A., HUI, P., WANG, R., AND MIN, A. W. Utilize signal traces from others? a crowdsourcing perspective of energy saving in cellular data communication. *IEEE Transactions on Mobile Computing* 14, 1 (2015), 194–207.
- [19] PATHAK, A., HU, Y. C., AND ZHANG, M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), ACM, pp. 29–42.
- [20] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTEHOFFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- [21] PELTONEN, E., LAGERSPETZ, E., NURMI, P., AND TARKOMA, S. Energy modeling of system settings: A crowdsourced approach. In *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on* (2015), IEEE, pp. 37–45.
- [22] SANZ, B., SANTOS, I., LAORDEN, C., UGARTE-PEDRERO, X., AND BRINGAS, P. G. On the automatic categorisation of android applications. In *2012 IEEE Consumer communications and networking conference (CCNC)* (2012), IEEE, pp. 149–153.
- [23] SHABTAI, A., FLEDEL, Y., AND ELOVICI, Y. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on* (2010), IEEE, pp. 329–333.
- [24] SHABTAI, A., KANONOV, U., ELOVICI, Y., GLEZER, C., AND WEISS, Y. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems* 38, 1 (2012), 161–190.
- [25] SRUBENSTEIN. Google play store apps permissions. <http://www.pewinternet.org/interactives/apps-permissions/>, Sep 2015.
- [26] STUTZ, D. Latex resources. <https://github.com/davidstutz/latex-resources>. Accessed: 2017-10-27.
- [27] TSAO, S.-L., KAO, C.-C., SUAT, I., KUO, Y., CHANG, Y.-H., AND YU, C.-K. Powermemo: a power profiling tool for mobile devices in an emulated wireless environment. In *System on Chip (SoC), 2012 International Symposium on* (2012), IEEE, pp. 1–5.
- [28] WEN, Y., WOLSKI, R., AND KRINTZ, C. Online prediction of battery lifetime for embedded and mobile devices. In *International Workshop on Power-Aware Computer Systems* (2003), Springer, pp. 57–72.
- [29] YANG, Z. Powertutor-a power monitor for android-based mobile platforms. *EECS, University of Michigan, retrieved September 2* (2012), 19.