What should mobile app developers do about machine learning and energy?

Andrea McIntosh[1] and Abram Hindle[2]

[1]Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada
[2]Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

Corresponding author:
Abram Hindle[2]

Email address: hindle1@ualberta.ca

ABSTRACT

Machine learning is a popular method of learning functions from data to represent and to classify sensor inputs, multimedia, emails, and calendar events. Smartphone applications have been integrating more and more intelligence in the form of machine learning. Machine learning functionality now appears on most smartphones as voice recognition, spell checking, word disambiguation, face recognition, translation, spatial reasoning, and even natural language summarization. Excited app developers who want to use machine learning on mobile devices face one serious constraint that they did not face on desktop computers or cloud virtual machines: the end-user's mobile device has limited battery life, thus computationally intensive tasks can harm end-user's phone availability by draining batteries of their stored energy. How can developers use machine learning and respect the limited battery life of mobile devices? Currently there are few guidelines for developers who want to employ machine learning on mobile devices yet are concerned about software energy consumption of their applications. In this paper we combine empirical measurements of many different machine learning algorithms with complexity theory to provide concrete and theoretically grounded recommendations to developers who want to employ machine learning on smartphones.

1 INTRODUCTION

Imagine we are in a hot new start-up and your app, which will be deployed to millions of phones, needs to take advantage of machine learning. Which machine learning algorithms should we employ to avoid sapping the energy of your customers' phones? Should we use neural networks since they are so popular, or should we stick to simpler models to save energy? In this work we address the questions of "how energy efficient are these machine learning algorithms?" and "which algorithms should we use on a mobile device?"

Machine learning is growing in popularity. Google in particular has made the results of machine learning available to the general public in terms of speech recognition (1), translation (2), computer vision, and search. Many machine learning implementations have been deployed to servers in the cloud, or data centers. But the popularity of mobile devices such as smartphones and tablets are causing a push toward mobile-apps that employ machine learning. One of the issues that mobile platforms face that servers and desktop computers do not, is that mobile platforms tend to rely on batteries for power and when the batteries are out of energy the mobile device is no longer available for use. This is different from data-centres that have machines on racks that face power limits and need constant cooling. Machine learning on mobile platforms is often out-sourced to the cloud, but the bandwidth to the cloud is quite limited so a lot of machine learning is pushed back to the mobile device itself. Some apps engage in computer vision, others learn from the textual and event based data on the phone to schedule appointments (3), and others link and organize documents (4).

If machine learning is pushed to mobile devices what should practitioners do about the software energy consumption of machine learning on their mobile devices? Surveys of developers and users have found that poor software energy consumption performance can lead to negative app-store reviews and poor user satisfaction (5; 6; 7). In this work we will empirically test, measure, and detail the costs and

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

trade-offs between machine learning performance and software energy consumption. We will show that there is no best algorithm but there are a wide range of trade-offs that one can make depending on the context that one is operating within. Furthermore not all energy consumption is CPU bound as some algorithms cost more in terms of memory-use than others that in a memory constrained environment can induce more energy consumption.

The contributions of this paper are:

- an empirical evaluation of the tradeoffs that machine learning algorithms make between accuracy and software energy consumption;
- concrete recommendations for choosing machine learning algorithms for use on mobile platforms;
- empirical testing and measurement of multiple machine learning contexts that demonstrate "one size does not fit all".

2 PRIOR WORK

Prior work relevant to this paper include machine learning, mobile devices, and software energy consumption research.

2.1 Software Energy Measurement

Software energy consumption is an up and coming field in software engineering and computer engineering. With the popularity of mobile devices and apps, more and more software engineering research is targeted to energy constrained platforms.

Energy consumption recommendations and guidelines for developers are popular avenues of research. Hasan *et al.* (8) and Pereira *et al.* (9) investigated the energy profiles of Java collections to help developers manually choose the right collection. Linares-Vásquez *et al.* (10) produced a methodology of finding energy consuming libraries and APIs in Android applications. Li *et al.* (11) discussed causes of energy consumption on Android.

Many researchers have investigated what developers know about software energy, which motivates this paper because most of the works conclude that developers are woefully ill-equipped to address software energy consumption concerns. Pinto *et al.* (12) and Malik *et al.* (13) sought questions developers were already asking. Pang *et al.* (5) surveyed developers to see what they understood about software energy consumption. Manotas *et al.* (14) went further and surveyed numerous industrial developers.

Recommenders quickly turn into optimizers that apply search techniques and find solutions to software energy consumption concerns. SEEDS from Manotas *et al.* (15) attempts to find the most energy efficient Java collections to use in a program for a particular context. GUI optimizations have also been approached using a search-based approach by Linares-Vásquez *et al.* (16). Bruce *et al.* (17) explicitly applied search-based software engineering techniques to mutate existing source code. Saborido *et al.* (18) use multi-objective heuristics to find optimal apps where energy is one dimension.

Measuring software energy consumption is another avenue of research. We used the GreenMiner (19) in this paper to measure software energy, but other researchers such as Banerjee *et al.* (20) have made their own measurement frameworks.

Numerous empirical studies exist about different aspects of software development juxtaposed against software energy consumption. Researchers such as Rasmussen *et al.* (21) and Gui *et al.* (22) have investigated the cost of advertisement on energy consumption. Chowdhury *et al.* (23) and Li *et al.* (24) benchmarked HTTP related energy concerns. Many researchers have suggested ranking and measuring apps by energy consumption (25; 26; 18).

A very popular area of research is the modelling of software energy consumption. Pathak *et al.* (27; 28) and Aggarwal *et al.* (29) used system-call based models. Chowdhury *et al.* (30) used count based models. Some tools attempt to diagnose the actual cause of software energy consumption in terms of the code (31).

2.2 Machine Learning on Mobile Platforms

Multiple frameworks exist that enable machine learning within mobile applications. As Android uses Java, any Java-based machine learning framework can easily be integrated into an Android application. For our tests, we used the Weka (32) and Neuroph (33) frameworks. Google Brain's TensorFlow machine learning library (1) is also intended to be portable to mobile and embedded devices.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

As a demo for an Android application, TensorFlow provides example code for an application that can classify what is being viewed in the phone's camera frame in real time. Similarly, the Google Translate mobile application can translate words being viewed through a phone's camera offline and in real-time using a trained convolutional neural net (2).

There are numerous cases of machine learning being used in apps. "Smart calendar" apps use machine learning to enhance calendar applications. Google Calendar Goals automatically schedules time for user-set personal goals, such as exercising three times a week, re-schedules these goals if a conflicting appointment is added, and learns the best times to schedule goals based on when the user completes or defers a goal (3). The Tempo app could pull and bundle data related to calendar events from the user's accounts — such as participant contact information, directions to the location, associated documents — and present them together in one organized entry (4).

Triposo is an app that provides travel recommendations and booking options to users. It uses machine learning to process websites and reviews, and combines the results with user preferences to make personalized recommendations (34). Weotta is an app that uses machine learning and natural language processing to provide event and activity recommendations to user queries (35).

2.3 Algorithms Used

We tested eight machine learning algorithms: *Naïve Bayes* (NB), *J48* (Weka's implementation of C4.5), *Sequential Minimal Optimization* (SMO) which is a support vector machine, *Logistic Regression* (LogReg), *Random Forest* (RF), *k-Nearest Neighbour* (IBk), ZeroR, and *MultiLayer Perceptron* (MLP) which is a neural network. All algorithm implementations except for MLP were from the Weka Java codebase. The MLP implementation, a neural network, is from the Neuroph framework.

ZeroR is a very simple classifier, that disregards any attribute information and always predicts the majority class of the training set. As such, ZeroR can provide the baseline accuracy for a dataset (36). For a dataset with n training instances, ZeroR will take $O(n)$ time to build a classifier as it needs to check the class value of each instance in order to find the most frequent class. However, it takes virtually no time, constant time $O(1)$, to classify.

Naïve Bayes is a type of Bayesian network that uses the simplifying assumptions that the predictive attributes are conditionally independent, and that there are no hidden attributes that influence predictions. With these simplifying assumptions, given a dataset with d attributes, n testing instances and m training instances, the Naïve Bayes classifier can perform training and testing in $O(dn)$ and $O(dm)$ time respectively (37). The Weka Naïve Bayes algorithm used for these tests is not updateable, although Weka also has an updateable implementation of Naïve Bayes.

J48 is Weka's implementation of the C4.5 decision tree algorithm (38). For a dataset with d attributes and n testing instances, C4.5 training has an algorithmic time complexity of $O(nd^2)$ (39).

SMO is an algorithm for training a Support Vector Machine (SVM) classifier, that breaks down the SVM quadratic programming optimization to simplify implementation, speed up computation, and save memory (40) (41). Platt found empirically that the training time of SMO ranges from $O(n)$ up to $O(n^{2.2})$ for n training instances (40). In Weka's implementation, datasets are automatically processed to replace missing values, normalize all attributes, and convert nominal attributes to binary ones.

Logistic Regression is a statistical machine learning algorithm. Using logistic regression with the Quasi-Newton method, a dataset with d attributes and n instances takes $O(d^2n+nd)$ time per iteration (42). For our tests logistic regression was set to iterate until convergence. Weka's implementation of the algorithm is slightly modified from the original Logistic Regression to handle instance weights.

Random Forest is an advanced tree classifier that grows multiple trees and allows them to vote for the best class (43). For a forest with L tress, n instances, and d attributes, theoretically the random forest will be constructed in $O(Ln^2d \cdot log(n))$ time, although practically the complexity is often closer to $O(Lnd \cdot log(n))$ (44).

IBk is an instance-based learner algorithm, that is similar to the k-nearest neighbour algorithm (45). For our tests, we classified instances based on the nearest three neighbours ($k = 3$). IBk is lazy when training, taking almost no time to create a model (46). However, for a dataset with d attributes and n instances, it takes $O(nd)$ to classify an instance (45).

MLP is a neural network implementation. For our tests, MLP used back-propagation learning and had only one hidden layer of neurons. The number of hidden neurons was fixed at 15 and the number of training epochs was fixed at 100. In general, for a dataset with n instances and a neural network with a

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 1. Size and type of datasets used in energy tests

Dataset	Description	Number of Attributes	Number of Instances	Number of Classes
MNIST	Image classifier – Integer attributes	785	5000	10
PGSQL	Text classification – Binary categorical attributes	2000	400	2
Mushroom	Classification – Categorical attributes	23	8124	2
Adult	Classification – Categorical, integer attributes	15	32561	2
Spambase	Text classification – Integer, real attributes	58	4601	2
Waveform	Numeric classification – Real attributes	22	5000	3
Pendigits	Image classifier – Integer attributes	17	10992	10

input neurons, b hidden neurons, and c output neurons, the network will take $O(nabc)$ time to train per epoch (47).

2.4 Datasets Used

We used seven existing datasets to test the machine-learning algorithms. The datasets chosen were of different sizes and datatypes, and represented different classification problems. We used our own text classification dataset (PGSQL) from our prior work (48; 49), the MNIST number classification dataset (50), and five datasets from the UCI archive (51) (Mushroom, Adult, Waveform, Spambase, and Pendigits). MNIST and Pendigits are image classification problems; PGSQL and Spambase are text classification problems; Adult and Waveform are numeric classification problems; and Mushroom is categorical classification.

Weka is designed to work with the ARFF file format. A version of the MNIST dataset already converted to the ARFF format was obtained (52) and used for the tests. The other datasets were converted to ARFF files using the Weka Explorer's conversion capabilities. For our tests, the size of the MNIST dataset was reduced to 5000 randomly selected instances. The size of the PGSQL dataset was also reduced from 640 instances with 23008 attributes to 400 instances with 2000 attributes, one of which was the class. The datasets are summarized in Table 1.

The MLP implementation we used from the Neuroph framework required datasets in CSV format. It also requires that numeric attributes be normalized to values between 0 and 1, nominal attributes and classes be represented as one-hot binary inputs, and instances with missing attribute or class values be removed beforehand. This processing and conversion to CSV was done using the Weka Explorer. As a result of converting categorical attributes to one-hot binary attributes, the number of input neurons for the Mushroom dataset became 111, and 104 for the Adult dataset.

A mirror of our datasets can be found at this url: `https://archive.org/details/mnist_test_reduced_5k`.

3 METHODOLOGY AND MEASUREMENTS

In this section we describe how we setup benchmarks for the machine learning algorithms and datasets. We also describe how we measured the energy consumption of the machine learning benchmarks.

3.1 Energy Measurement with GreenMiner

Energy and power measurements were collected using the GreenMiner energy-measurement framework. This framework uses hardware-instrumented Android smartphones to physically measure the energy consumption and power use of apps running on the phones (19). It automatically runs submitted tests and uploads the results to a central webservice. Before each test is run, the application APK (Android package) is installed on the phone, required data is uploaded onto the SD card, and phone settings such as screen brightness, and screen timeout are set as required. After each test the application is uninstalled, the data is deleted from the SD card, settings are restored to previous values, and data generated during the tests such as log-files are pulled from the phones to be uploaded to the web service and then deleted from the phone, so that the next test can begin with a clean environment. Tests run for a set duration, and testers can split the test's energy measurements into partitions of varying duration to capture the energy and power use of different phases of app execution. Such a phase could be reading the data or training the

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

model. The GreenMiner measures and reports information about the test run including energy use, power use, and runtimes for both the entire test duration and over each tester-specified partition. An example of an energy profile for a cross-validated Naïve Bayes test displayed on GreenMiner's web interface is shown in Figure 1.

3.2 Measurement Process

To test machine learning algorithms on the GreenMiner phones, two Android apps were created. An app was created to run Weka machine learning algorithms, based on an existing modification of the Weka codebase that can run on Android.[1] A second app was created to test a MultiLayer Perceptron neural net algorithm, using the Neuroph framework. Both apps ran the same datasets.

Tests of the different algorithms and datasets were written as Android `InstrumentationTestCases`, with the phases of evaluating an algorithm (reading data, training the model, validating the model) written as separate tests. The different tests were initiated by pressing buttons, and data was transferred between different test methods via a singleton object. To keep the screen energy consumption of the apps constant, the screens were almost completely black, with some small grey text on the buttons for debugging purposes. Both the Weka and the Neuroph apps had exactly the same user interface.

Tests were created for eight different machine learning algorithms to evaluate seven different datasets. Separate tests methods were written to perform two different types of evaluation. For each algorithm two tests were written to train on 50% of the data and then test on the other 50%. Two more tests were written to train and test on the whole dataset using 10-fold cross validation. Each train/test evaluation pair was run separately on the GreenMiner.

Each test method was invoked in turn by pressing a button on the app's interface once the previous method had completed. The GreenMiner framework cannot automatically detect when a test method has completed, because it runs uninstrumented, so in order to invoke the next method initial timing test runs were performed to determine appropriate delays to add to the GreenMiner scripts. Each algorithm-dataset-validation combination was run at least 10 times on the GreenMiner so that their results could be averaged and to allow for enough statistical power to determine an effect. Some combinations, such as random forest on the MNIST dataset with cross validation, ran out of memory when evaluating on the phones, and so are not included in our results.

The GreenMiner collects the energy consumption measurements and power measurements of each test method. The results of all successful test runs were compiled and compared. For comparisons, the training and testing phases of 50% split evaluation are combined, and are compared against the energy for cross-validating with 10-folds, that includes training and testing each fold. Energy consumption measurements are compared to determine which algorithms will require the most or least energy to evaluate on each dataset. Power usages are compared to determine if some algorithms are more energy-hungry, independent of how long it takes them to evaluate.

The correctness of the Weka algorithms was gathered from the Weka 3.8 desktop application, based on performing 10-fold cross validation. The total root-mean-squared errors (RMSE) of the MLP algorithm were gathered from NeurophStudio. The average accuracies of an algorithm over all datasets were compared to determine which algorithms were generally the most or least accurate. The accuracy for Logistic Regression could not be calculated for the Adult dataset because the desktop Weka application ran out of memory.

Statistical significance testing was executed using a Student's t-test as energy measurement data typically is normally distributed. Anders-Darling tests confirmed normality in most cases. We addressed multiple hypotheses and comparisons by applying Bonferroni correction with an initial alpha (α) of 0.05.

4 ENERGY PROFILING RESULTS

We profiled the energy and power use of eight machine learning algorithms, and compared how they varied with datasets of different sizes. We compared how eight machine-learning algorithms used power and energy when applied to datasets of different sizes. We asked four research questions:

RQ1: Can we identify the best performing algorithm in terms of energy?
RQ2: Can we identify the best performing algorithm in terms of power?
RQ3: Can we identify the best performing algorithm in terms of accuracy?
RQ4: Can we identify the best performing algorithm for training/testing in terms of energy?

[1] Weka for Android `https://github.com/rjmarsan/Weka-for-Android`

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 2. Average ranking of each algorithm from lowest to highest energy consumption

Sorted Algorithm	Rank – 50%	Sorted Algorithm	Rank – 10-CV
ZeroR	1	ZeroR	1
NB	2.57	NB	2
J48	3.57	J48	3.86
SMO	3.86	SMO	4.43
LogReg	5.43	LogReg	5
MLP	6.29	IBk	5.29
IBk	6.57	RF	7.14
RF	6.71	MLP	7.29

4.1 RQ1: Can we identify the best performing algorithm in terms of energy?

Which algorithms are more energy efficient? Figure 2 shows the energy used to train and test the algorithms on a 50% split of each dataset. Figure 3 shows the energy used to perform 10-fold cross validation on the algorithms for each dataset. Note that some algorithms could not be evaluated on some datasets, and so not all algorithm-dataset combinations are shown in the figures.

Generally, energy consumption increases with increasing dataset size, however these increases typically do not strictly follow a clear trend. One reason for deviations could be related to memory cache; spikes in energy consumption could be due to the memory cache exhaustion for that particular dataset.

Figure 2 shows that other than ZeroR, Naïve Bayes and J48 tend to have the lowest energy consumption for 50%-split. SMO also has good energy performance for most datasets except for the Adult dataset. Figure 3 shows that Naïve Bayes is consistently consumes the nearly the least energy for cross validation, and J48 is one of the highest energy users for smaller dataset sizes, but one of the lower energy consumers for larger datasets.

The overall rankings of the algorithms' energy use were determined by assigning a rank value to each algorithm for each dataset, with 1 using the least energy and 8 using the most. The rankings for each dataset were then summed, and divided by the number of datasets. Table 2 shows that ZeroR always uses the least amount of energy, followed by Naïve Bayes and J48. There were some deviations in the rankings of each algorithm on a dataset between cross-validation and 50% split. The order of average rankings for each evaluation method had high correlation of 0.93.

The energy use of the algorithms were compared using a pairwise t-test to determine if the energy differences are statiscally significant for an alpha of 0.05. For the combined training and testing energies of 50% split, all algorithms had signifcantly different energy consumptions except for NB vs J48, J48 vs LogReg, J48 vs RF, SMO vs IBk, SMO vs MLP, and IBk vs MLP. For cross validation, all algorithms had significantly different energy consumptions except for J48 vs LogReg, J48 vs IBk, LogReg vs IBk, LogReg vs RF, IBk vs RF, and MLP vs RF.

4.2 RQ2: Can we identify the best performing algorithm in terms of power?

Figure 4 shows the average power use to train and test the algorithms on a 50% split of each dataset. Figure 5 shows the average power use of each algorithm to perform 10-fold cross validation. Note that some algorithms could not be evaluated on some datasets, and so not all algorithm-dataset combinations are shown in the figures.

Figures 4 and 5 show that the power use of all algorithms are similar. Table 3 shows the average rankings for the algorithms are less evenly-spread between 1 and 8, indicating that the rank of an algorithm's power use varies more from dataset to dataset. Additionally, the rankings of algorithms between 50% split and cross validation are not as well-correlated as the energy rankings, with a Spearman's rank correlation rho value of 0.62. However, overall the algorithms' power rankings are similar to the energy rankings, with ZeroR and Naïve Bayes consistently having the lowest power consumption.

The power use of the algorithms were compared using a pairwise t-test to determine if the power use differences are statiscally significant for an alpha of 0.05. For the combined training and testing energies of 50% split, all algorithms had signifcantly different power consumptions except for J48 vs MLP, SMO vs LogReg, SMO vs RF, SMO vs IBk, LogReg vs IBk, and RF vs IBk. For cross validation, all algorithms

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 3. Average ranking of each algorithm from lowest to highest power use

Sorted Algorithm	Rank – 50%	Sorted Algorithm	Rank – 10-CV
ZeroR	1.43	ZeroR	1.14
NB	3.14	NB	2.86
MLP	3.57	LogReg	3.71
J48	4.43	J48	4.29
SMO	4.71	MLP	5
IBk	5.86	IBk	5.71
RF	6.14	SMO	6.29
LogReg	6.71	RF	7

Table 4. Average algorithmic accuracies ordered based on percentage of correctly classified instances, kappa statistic, and Root Mean Squared Error

Accuracy	Algorithm	% Correct	Algorithm	Kappa	Algorithm	RMSE
Most	MLP	95.66%	MLP	0.9293	MLP	0.08
	Random Forest	90.32%	SMO	0.7488	Random Forest	0.21
	SMO	90.13%	Random Forest	0.7211	IBk	0.21
	IBk	88.32%	IBk	0.7194	LogReg	0.25
	LogReg	87.08%	LogReg	0.7087	J48	0.25
	J48	85.73%	J48	0.6911	SMO	0.29
	Naïve Bayes	81.97%	Naïve Bayes	0.6332	Naïve Bayes	0.32
Least	ZeroR	46.36%	ZeroR	0.0000	ZeroR	0.41

had significantly different power consumptions except for NB vs LogReg, NB vs MLP, NB vs RF, J48 vs IBk, SMO vs IBk, LogReg vs MLP, LogReg vs RF, and MLP vs RF.

4.3 RQ3: Can we identify the best performing algorithm in terms of accuracy?

Algorithmic accuracy is determined based on the percentage of correctly classified instances and on the kappa statistic. Kappa measures agreement between the predicted and the true class. As different algorithms sometimes had the same accuracy for a dataset, rather than ranking algorithmic accuracy for each dataset — which would result in ties — the average accuracy of each dataset was calculated. As the accuracy for Logistic Regression could not be calculated for the Adult dataset, the average for Logistic Regression was taken over only 6 values, while the other algorithms were calculated over 7. Table 4 shows the algorithms ordered in terms of both measures of accuracy.

Weka outputs predicted classes, and also provided a calculation of the root mean squared error (RMSE) of the predictions. Neuroph outputs the probabilities of each class. The outputs of the five datasets that could run on GreenMiner with cross validation (PGSQL, Mushroom, Waveform, Spam, and Pen) were normailzed using softmax, and the highest normalized probability was taken as the predicted class. From this, the accuracies and kappa statics for MLP on each dataset were computed in R. The total RMSE of MLP on each dataset was obtained from NeurophStudio. The average RMSE of each algorithm over all datasets is included in Table 4.

Table 4 shows the most accurate Weka algorithms are Random Forest and SMO; their percentage of correctly classified instances are very close, with Random Forest being about 0.2% higher. Yet SMO had a slightly better kappa statistic implying its classifications are more balanced. Overall, MLP is clearly the most accurate algorithm. It has significantly higher average classification accuracy and kappa statistic than the next-best algorithms, and the lowest RMSE.

4.4 RQ4: Can we identify the best performing algorithm for training/testing in terms of energy?

Figure 6 compares the average energy to train and test each algorithm over all datasets with 50% split. Lazy algorithms such as IBk were the most efficient for training, followed by Naïve Bayes. For

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 5. Spearman rank correlation rho value for 50% split energy use and CPU use between algorithms classifying a dataset

Dataset	User Time	System Time	Idle Time	IO Wait Time	Number of Interrupts	Context Switches	Processes
Adult	1.00	0.57	1.00	0.07	0.96	0.79	0.85
MNIST	1.00	0.61	1.00	0.04	0.96	0.82	0.93
Mushroom	1.00	0.76	0.90	0.52	0.95	0.86	0.64
Pendigits	0.98	0.36	1.00	0.57	0.95	0.74	0.83
PGSQL	1.00	0.19	0.98	0.17	0.76	0.12	0.81
Spambase	1.00	0.00	0.98	0.45	0.79	0.07	0.50
Waveform	1.00	0.14	0.93	0.19	0.67	0.33	0.95

Table 6. Spearman rank correlation rho value for CV energy use and CPU use between algorithms classifying a dataset

Dataset	User Time	System Time	Idle Time	IO Wait Time	Number of Interrupts	Number of Context Switches	Number of Processes
Adult	1.00	0.90	1.00	0.30	1.00	0.90	1.00
MNIST	1.00	1.00	1.00	0.50	1.00	1.00	1.00
Mushroom	1.00	0.88	1.00	0.71	0.95	0.83	0.93
Pendigits	1.00	0.76	1.00	0.33	0.98	0.81	0.98
PGSQL	1.00	0.57	1.00	0.21	0.96	0.75	0.93
Spambase	1.00	0.21	1.00	0.25	0.86	0.57	0.93
Waveform	1.00	0.36	1.00	0.18	0.86	0.57	0.96

evaluation/classification other than ZeroR, J48 was quite efficient to classify data in terms of energy. For both training and test combined Naïve Bayes performed well.

5 CAUSES OF ENERGY DIFFERENCES

5.1 Is energy use related to the CPU usage of an algorithm?

Before and after running a test, the phone's `/proc/stat` file is collected to gather information about the phone's CPU time and processes. The difference between the two measurements is used to determine the CPU time and resource usage of a test. These results are compared to determine how an algorithm's CPU usage is related to its energy usage.

When comparing the results from 50%-split tests, energy use was strongly correlated to user time and idle time for all datasets. Table 5 shows that energy consumption was not strongly correlated to system time usage or IO wait time for most datasets. Energy was strongly correlated to the number of interrupts for most datasets, except for PGSQL and Waveform, where it was only moderately correlated. For other CPU use measurements, the strength of correlation to energy usage varied widely between datasets. The results were similar for cross-validation.

In general, the correlations between energy use and CPU use were stronger for cross validation. It should be noted that the Adult and MNIST could not be evaluated by many algorithms on the phones because they ran out of memory. Thus, there are fewer energy results to compare for these datasets.

For the 10-fold results, energy use was strongly correlated to user time, idle time, and number of processes. The number of interrupts was also well-correlated to energy use for all datasets. IO wait time was not strongly correlated to energy use, and, excluding the Adult and MNIST values, system time was generally not strongly correlated to energy use for any dataset.

The number of processes did not significantly increase between 50% split evaluation compared to cross validation. On average, over all datasets and algorithms, only 1.2 times as many processes were created for cross validation as compared to 50% split. In contrast, on average, 10-fold evaluation used 7.0 times more idle time, and 10.5 times as much user time.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 7. Average memory usage of each algorithm over all datasets

Algorithm	Number of Concurrent GC	GC Concurrent Time (ms)	Number of GC for Alloc	GC for Alloc Time (ms)	Times Grown	Used (Bytes)	Allocated (Bytes)
IBk	148	4853	79	3449	34	12647	21148
J48	332	22650	27	1268	9	13853	18139
LogReg	942	69496	1592	86693	121	31019	35258
MLP	698	24260	286	16671	1	6966	12022
NB	668	32272	16	573	4	9818	12914
RF	957	122458	244	18323	74	28504	50757
SMO	328	13448	381	15336	226	28189	37138
ZeroR	135	3674	6	189	1	8989	11348

5.2 Is energy use related to the memory use of an algorithm?

Android's Dalvik VM automatically logs information about heap use and garbage collection (GC). These logs were collected for the algorithms and datasets using Android's logcat tool. These logs have the number of kilobytes allocated for and used on the heap, the number of times the app's heap size was grown, the number of concurrent GCs performed when the heap grows too large, the number of GCs performed when the heap is too full to allocate required memory, and the total time taken to perform these GCs, could be parsed and compared. The average results for each algorithm performing 10-fold cross validation over all datasets are shown in Table 7.

Logistic Regression and Random Forest used the most memory on the heap and performed the most concurrent garbage collections. Overall, they are the most inefficient in terms of memory use. It should also be noted that Random Forest's performance was most affected by memory, as five datasets could not be evaluated with 10-fold cross validation on the phones as they ran out of memory or had a stack overflow occur. Excluding both MLP and ZeroR, Naïve Bayes, J48, and IBk performed the fewest garbage collections to make space for allocations, grew their heap the fewest number of times, and used the least amount of heap space. Random Forest and Logistic Regression were both large energy users, while Naïve Bayes and J48 were the lowest energy users, so for these algorithms their memory use seems related to their energy use. However, IBk was one of the most memory-efficient, but the second-highest energy consumer, so memory use alone cannot account for memory efficiency. Additionally, MLP, which was implemented with the Neuroph framework rather than Weka, was very memory efficient despite being the highest energy user with cross validation. Excluding ZeroR, MLP used and allocated the least amount of heap space, and grew its heap the fewest number of times. However, it performed the third-most GCs, so it is may be reducing its memory requirements by performing more frequent memory clean-ups.

The memory use of the Weka-implemented algorithms, not MLP, was compared to energy use, and the Spearman's correlation rho estimates are shown in Table 8. Table 8 shows that energy use is not consistently well-correlated to memory use. Generally energy use was most strongly correlated to the maximum heap space used in a test and the maximum heap space allocated in a test. Spambase and Waveform datasets generally showed weak correlations between their energy and memory use.

When the MLP memory usage data is added to the comparison most of the correlations were unchanged or became weaker as, exhibited by Table 9, although some correlations — particularly for the Waveform dataset — became stronger.

5.3 Is energy use related to the methods called by an algorithm?

Method traces for algorithms with different datasets were generated using Android's Dalvik Debug Monitor Server (DDMS) and dmtracedump tools. The method traces were generated by sampling every millisecond. The methods called by each algorithm are compared, and the total number of CPU cycles and total number of method calls made are correlated to energy use.

The total number of method calls is strongly correlated to the energy use of each algorithm on a dataset, with algorithms making more method calls using more energy. All datasets had rho estimates of 0.9 or better. Similarly, the number of CPU cycles elapsed during execution also had a rho estimate of 0.9 or better for all datasets when correlated to energy use.

Additionally, algorithms that used more energy, such as MLP or Random Forest, called costly methods

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 8. Spearman's rank correlation rho value for 10-fold energy use and memory use between Weka-implemented algorithms classifying a dataset

Dataset	GC Concurrent	GC Concurrent (ms)	GC for Alloc	GC for Alloc (ms)	Grow	Used	Allocated
Adult	0.40	0.70	0.90	0.90	0.87	0.70	0.90
MNIST	0.50	0.50	1.00	1.00	1.00	1.00	1.00
Mush	0.75	0.75	0.64	0.64	0.26	0.96	0.96
Pen	0.68	0.68	0.79	0.82	0.71	0.86	0.86
PGSQL	0.71	0.71	0.77	0.83	0.06	0.66	0.66
Spam	0.49	0.49	0.49	0.60	0.60	0.60	0.60
Wave	0.14	0.31	0.60	0.60	0.60	0.60	0.66

Table 9. Spearman's rank correlation rho value for CV energy use and memory use between all algorithms classifying a dataset

Dataset	GC Concurrent	GC Concurrent (ms)	GC for Alloc	GC for Alloc (ms)	Grow	Used	Allocated
Adult	0.4	0.7	0.9	0.9	0.87	0.7	0.9
MNIST	0.5	0.5	1	1	1	1	1
Mush	0.69	0.69	0.42	0.42	0.19	0.74	0.74
Pen	0.79	0.76	0.69	0.74	0.34	0.67	0.67
PGSQL	0.36	0.57	0.86	0.86	-0.19	0.5	0.5
Spam	0.65	0.65	0.47	0.47	0.44	0.76	0.68
Wave	0.54	0.65	0.68	0.68	0.72	0.68	0.94

many times. For the applicable datasets Random Forest was able to perform cross validation to completion on, the method invoked the most number of times by the algorithm was Weka's QuickSort. Naïve Bayes and J48 also invoked QuickSort, but significantly fewer times per dataset: Random Forest called QuickSort 9 to 41 times as often as often as J48 did, and 69 to 83 times as often as Naïve Bayes. QuickSort was never used on the Mushroom dataset with any algorithm as it only has categorical attributes. MLP called methods to update weights with backpropagation calculations the most. Logistic regression, another high energy-user, frequently calls methods to evaluate the model's gradient vector and to perform exponentiation.

5.4 Is energy use related to algorithmic complexity?

To determine the correlation between algorithmic complexity and energy usage, the relevant statistics of each dataset, including number of attributes, and number of instances, were substituted into the algorithmic time complexity formulas for training each learner. For IBk, which has a constant time complexity, the cost was set to the constant 100000 for each dataset. For SMO, which was empirically determined to have a time complexity between $O(n)$ up to $O(n^{2.2})$ for n training instances (40), a time complexity of $O(n^2)$ was used. The rho values for the Spearman correlations between these computed numeric complexities and the energy required to train each algorithm on a dataset are shown in Table 10.

The curves of these complexity functions were then tuned by a single coefficient for a better fit. J48 was multiplied by a factor of 5, Logistic Regression by 75, Random Forest by 10, and MLP by 100. The new rho estimates from these tuned curves are shown in Table 11.

Table 10. Spearman correlation rho estimates between algorithmic complexity and energy consumption when training model

	PGSQL	MNIST	Mush	Adult	Wave	Spam	Pen
50%	0.81	0.82	0.83	1.00	0.81	0.76	0.90
10-CV	0.86	1.00	0.83	1.00	0.75	0.64	0.93

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Table 11. Spearman correlation rho estimates between algorithmic complexity tuned with constant factors and energy consumption when training model

	PGSQL	MNIST	Mush	Adult	Wave	Spam	Pen
50%	0.81	0.96	0.83	0.96	0.90	0.93	0.93
10-CV	0.86	1.00	0.83	1.00	0.89	0.89	0.98

5.5 Analysis

Hasan *et al.* (8) found that the power use of different collection classes was similar, and that energy consumption seemed to increase at the same rate as program runtimes, indicating that programs that use more energy do so because they do more work in the extra time it takes them to run. Our results agree with this.

While the energy consumptions of different algorithms could differ significantly, the algorithms tended to have similar power use. This is likely because the processes are primarily CPU bound. We found that energy use was positively correlated to both runtime complexity, and the user and idle CPU time taken by an algorithm. Further, energy use was positively correlated to the number of methods called by an algorithm during execution, indicating that algorithms that use more energy to evaluate a dataset both take longer and call more methods, thus doing more work. Algorithms and datasets that invoked garbage collection more typically took longer and consumed more energy.

6 EVALUATING MACHINE LEARNING CHOICES ON MOBILE DEVICES

In this section we provide guidance to app developers who seek to use machine learning within their mobile-apps. Developers should decide if they need to train machine learners or if they can simply share a trained model with their mobile-app. Developers should also consider the effect that the number of attributes have on energy consumption. Furthermore developers should consider how much energy consumption they are willing to allow for versus the accuracy or agreement they want to achieve.

6.1 What are the best algorithms to use for models that do not need updating?

The Google Translate application uses a convolutional neural net that was trained on a carefully selected dataset, and then deployed in the application (2).

J48, SMO, Logistic Regression, and MLP all have significantly higher training costs than classifying costs. Thus, these algorithms would be ideal for implementations where the model could be trained ahead of time, and not updated after release for classification in the application. J48, Logistic Regression and SMO are Pareto optimal choices based on our limited evaluation, depicted in Figure 7.

6.2 What are the best algorithms to use for models that need updating?

If the model must be trained or re-trained on the phone, Naïve Bayes is the best algorithm to use to limit energy use, as it has the lowest energy use overall and has the same time complexity for training as for classifying [8]. The IBk classifier is trivial to update, making updating fast and low-energy, but it is slow and energy-intensive to classify and it is one of the worst energy consumers for classification.

6.3 What are the best algorithms to use to minimize energy consumption?

Excluding ZeroR, Naïve Bayes used the least amount of energy on average for training and testing. J48 was also energy efficient, being the next-lowest energy user on average, after Naïve Bayes. Thus, Naïve Bayes and J48 are the best algorithms to use for applications trying to reduce energy use. For 50% split training and testing Naïve Bayes was the lowest energy consumer on average, but was the second-lowest energy consumer for some datasets. For cross-validation, Naïve Bayes was the lowest energy consumer across all datasets. This suggests that Naïve Bayes' energy performance will scale well over time.

Naïve Bayes is recommended over J48 in terms of energy use if the model must be trained as well as evaluated by the app. If the model can be pre-trained, J48 will likely use less energy and be faster to validate than Naïve Bayes, but Naïve Bayes can train models faster and with less energy than J48.

6.4 What are the best algorithms to use to maximize accuracy?

Of the Weka algorithms, Random Forest and SMO were the best classifiers overall, with Random Forest having the highest average accuracy and SMO having the highest average kappa statistic, making these

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

the best algorithms to use to obtain correct results. Random Forest was also the highest average energy user on 50% split datasets, and the second highest for 10-fold evaluation. SMO was less energy-hungry overall and dominated RF.

MLP had the highest average accuracy overall, with an average classification accuracy of over 95% and an average kappa of over 0.92. On some datasets it was able to achieve RMSEs smaller than 0.0001, suggesting potential overfitting. MLP could likely achieve even higher accuracies if optimized. To standardize the tests, all our MLP networks had the same number of hidden neurons (15), learning rate (0.2), and fixed number of training epochs (100) regardless of input size or type. Tuning these parameters for each dataset could likely improve prediction accuracies. For example, the Spambase dataset had the highest error, with a classification total mean square error of 0.37 with the test parameters, but using a learning rate of 0.1 and 1000 training epochs, the total mean square error could be reduced to 0.31. However, tuning these parameters would likely also affect energy consumption of the network.

6.5 What are the best algorithms for datasets with many attributes?

Energy consumption is strongly-correlated to algorithmic time complexity. Thus, it is not surprising that the algorithms with the lowest energy use on datasets with large numbers of attributes (PGSQL, MNIST, Spambase) also have algorithmic complexities that have a low dependence on the number of attributes. SMO had low energy use on the PGSQL and Spambase datasets, especially with 50% split evaluation. Naïve Bayes, which has a linear dependence on the number of attributes, also performs well on these datasets.

6.6 What algorithms dominate in terms of energy versus accuracy?

Figure 7 shows a clear dominating Pareto front of machine learners that are "optimal" for energy consumption or accuracy measured in Kappa score. Clear dominators in order of Kappa score versus energy are ZeroR, J48, Logistic Regression and support vector machines (SMO). These candidates make sense because they are effectively small functions (logistic regression and SMO) or conditions (J48) that are quick to evaluate. For training, ZeroR, IBk and SMO dominate as IBk's lazy training beats Naïve Bayes. Ignoring IBk, the training dominators are in order of Kappa are: ZeroR, Naïve Bayes, J48, logistic regression, RF, and SMO.

7 THREATS TO VALIDITY

Construct validity is threatened by our choice of experiments, machine learning algorithms, and data sets. We tried to control for attribution errors by having a constrained environment that was very similar for every run.

Internal validity is threatened by selection bias of datasets and algorithms, as well the use of two machine learning frameworks. The consistency of the measuring framework could affect internal validity.

External validity is threatened by the limited number of machine learning algorithms evaluated. We could apply more and furthermore we are limiting ourselves to only two machine learning frameworks. Some frameworks could have better energy efficiency or run-times. We hope that a lot of the external validity can be addressed with the theoretical run-time estimates provided by complexity estimates.

8 CONCLUSIONS

We conclude that machine learning can be used in an energy effecient manner on mobile devices such as smartphones. Currently we would not recommend training neural nets on mobile devices, however evaluation with neural networks on mobile devices is quite successful (1; 2).

We observed that many machine learning algorithms cost more to train them to evaluate. Many of the issues with applying these machine-learning algorithms can be addressed by offloading the training to the cloud — which we recommend for logistic regression, support vector machines, and neural networks.

Depending on the context and the need for updates, a lazy trainer, such as nearest neighbours, with expensive evaluation could make more sense than an algorithm with relatively good performance balance between training and evaluation. One needs to balance how much evaluation versus how much training one needs to do. Constant evaluation implies one needs a cheap evaluator whereas constant updates and changing signals implies one need an algorithm that is cheap to train, such as Naïve Bayes or nearest neighbours.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

Dominating algorithms for only evaluation include Support Vector Machine, Logistic Regression and J48. Support Vector Machines, Random Forest, and Neural Nets (MLP) performed the best in terms of accuracy but with poor energy efficiency for training. Naïve Bayes was balanced and offered good accuracy compared with its training energy efficiency but suffers from high evaluation energy costs. Some algorithms did not fare very well for training such as logistic regression that requires lots of memory and CPU and had middle-ground accuracy without the ability to update easily.

Thus mobile app developers need to be aware of the trade-offs between different machine learning algorithms. We conclude that neural networks have good performance but suffer from poor energy efficiency in terms of both training and evaluation. Perhaps fixed-point or binarized neural networks as suggested by Courbariaux et al. (53) will enable the training of neural networks and deep learning on mobile devices.

Future work would be to integrate smart search techniques to emulate the SEEDS approach (15) of choosing machine learning algorithms given domain context and constraints. Thus, recommender systems could be built that could analyze the problem and make the best suggestion based upon empirical and theoretical constraints and measurements. Future work can also include accounting for more neural-net architectures, more learners, and more data-sets.

REFERENCES

[1] TensorFlow, "Mobile tensorflow," https://www.tensorflow.org/mobile.html, 2016.

[2] O. Good, "How google translate squeezes deep learning onto a phone," Google Research Blog https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html, July 2015.

[3] Google, "Find time for your goals with google calendar," Google Blog https://googleblog.blogspot.ca/2016/04/find-time-goals-google-calendar.html, July 2016.

[4] Christina Bonnington, "Your smartphone gains a mind of its own," Conde Nast http://www.wired.com/2013/07/ai-apps-trend/, July 2013.

[5] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" *IEEE Software*, pp. 83–89, 2015. [Online]. Available: http://softwareprocess.ca/pubs/pang2015IEEESoftware.pdf

[6] Hern and a. Alex, "Smartphone now most popular way to browse internet – ofcom report," https://www.theguardian.com/technology/2015/aug/06/smartphones-most-popular-way-to-browse-internet-ofcom/, 2015, (last accessed: 2016-Jul-29).

[7] V. Woollaston, "Customers really want better battery life," http://www.dailymail.co.uk/sciencetech/article-2715860/Mobile-phone-customers-really-want-better-battery-life-waterproof-screens-poll-reveals.html, uSwitch.com, 2014, (last accessed: 2015-APR-22).

[8] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *International Conference on Software Engineering (ICSE 2016)*, 2016, inproceedings, pp. 225–236. [Online]. Available: http://softwareprocess.ca/pubs/hasan2016ICSE-Energy-Profiles-of-Java-Collections-Classes.pdf

[9] R. Pereira, M. Couto, J. a. Saraiva, J. Cunha, and J. a. P. Fernandes, "The influence of the java collection framework on overall energy consumption," in *Proceedings of the 5th International Workshop on Green and Sustainable Software*, ser. GREENS '16, 2016, pp. 15–21.

[10] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597085

[11] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An empirical study of the energy consumption of android applications," in *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 2014, pp. 121–130. [Online]. Available: http://dx.doi.org/10.1109/ICSME.2014.34

[12] G. Pinto, F. Castor, and Y. D. Liu, "Mining Questions About Software Energy Consumption," in *MSR 2014*, 2014, pp. 22–31. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597110

[13] H. Malik, P. Zhao, and M. Godfrey, "Going green: An exploratory analysis of energy-related questions," in *Proceedings of the 12th Working Conference on Mining Software Repositories*,

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

ser. MSR ’15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 418–421. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820576

[14] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, “An empirical study of practitioners’ perspectives on green software engineering,” in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE ’16. New York, NY, USA: ACM, 2016, pp. 237–248. [Online]. Available: http://doi.acm.org.login.ezproxy.library.ualberta.ca/10.1145/2884781.2884810

[15] I. Manotas, L. Pollock, and J. Clause, “Seeds: A software engineer’s energy-optimization decision support framework,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 503–514. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568297

[16] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, “Optimizing energy consumption of guis in android apps: A multi-objective approach,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786847

[17] B. R. Bruce, J. Petke, and M. Harman, “Reducing energy consumption using genetic improvement,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’15. New York, NY, USA: ACM, 2015, pp. 1327–1334. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754752

[18] R. Saborido, G. Beltrame, F. Khomh, E. Alba, and G. Antoniol, “Optimizing user experience in choosing android applications,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 438–448.

[19] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. Campbell, , and S. Romansky, “Greenminer: a hardware based mining software repositories software energy consumption framework,” in *International Working Conference on Mining Software Repositories (MSR 2014)*, 2014, inproceedings, pp. 12–21. [Online]. Available: http://softwareprocess.ca/pubs/hindle2014MSR-greenminer.pdf

[20] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, “Detecting energy bugs and hotspots in mobile apps,” in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 588–598.

[21] K. Rasmussen, A. Wilson, , and A. Hindle, “Green mining: energy consumption of advertisement blocking methods,” in *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS 2014)*, 2014, inproceedings, pp. 38–45. [Online]. Available: http://softwareprocess.ca/pubs/rasmussen2014GREENS-adblock.pdf

[22] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond, “Truth in advertising: The hidden cost of mobile ads for software developers,” in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. IEEE, 2015, pp. 100–110. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2015.32

[23] S. Chowdhury, V. Sapra, and A. Hindle, “Client-side energy efficiency of http/2 for web and mobile app developers,” in *23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, 2016, inproceedings, pp. 529–540. [Online]. Available: http://softwareprocess.ca/pubs/chowdhury2016SANER-http2.pdf

[24] D. Li, Y. Lyu, J. Gui, and W. G. J. Halfond, “Automated energy optimization of http requests for mobile applications,” in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE ’16. New York, NY, USA: ACM, 2016, pp. 249–260. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884867

[25] Z. Chenlei, A. Hindle, , and D. M. German, “The impact of user choice on energy consumption,” *IEEE Software*, pp. 69–75, 2014. [Online]. Available: http://softwareprocess.ca/pubs/zhang2014IEEESoftware-user-choice.pdf

[26] R. Jabbarvand, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann, “Ecodroid: an approach for energy-based ranking of android apps,” in *Proceedings of the Fourth International Workshop on Green and Sustainable Software*. IEEE Press, 2015, pp. 8–14.

[27] A. Pathak, Y. C. Hu, and M. Zhang, “Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices,” in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X, 2011, pp. 5:1–5:6.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

[28] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, “Fine-grained Power Modeling for Smartphones Using System Call Tracing,” in *EuroSys ’11*, Salzburg, Austria, April 2011, pp. 153–168. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966460

[29] K. Aggarwal, A. Hindle, and E. Stroulia., “Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption,” in *International Conference on Software Maintenance and Evolution (ICSME 2015)*, 2015, inproceedings, pp. 311–320. [Online]. Available: http://softwareprocess.ca/pubs/aggarwal2015ICSME-greenadvisor.pdf

[30] S. A. Chowdhury and A. Hindle, “Greenoracle: Estimating software energy consumption with energy measurement corpora,” in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR ’16. New York, NY, USA: ACM, 2016, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901763

[31] K. Aggarwal, A. Hindle, and E. Stroulia, “Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption,” in *31st IEEE International Conference on Software Maintenance and Evolution*. IEEE Computer Society, 2015.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: An update,” *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[33] Z. Sevarac, I. Goloskokovic, J. Tait, L. Carter-Greaves, A. Morgan, and V. Steinhauer, “Neuroph: Java neural network framework,” http://neuroph.sourceforge.net/, 2016.

[34] Triposo, “Triposo,” https://www.triposo.com/, 2016.

[35] Weotta, “About weotta,” http://www.weotta.com/about, 2016.

[36] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 3rd ed. Morgan Kaufmann, 2011.

[37] G. H. John and P. Langley, “Estimating continuous distributions in bayesian classifiers,” in *Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[38] E. Frank, “Class j48,” http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html, 2009.

[39] Su, Jiang and Zhang, Harry, “A fast decision tree learning algorithm,” in *American Association for Artificial Intelligence*, vol. 6, 2006, pp. 500–505.

[40] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” in *Advances in Kernel Methods - Support Vector Learning*, B. Schoelkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998. [Online]. Available: http://research.microsoft.com/~jplatt/smo.html

[41] S.S. Keerthi and S.K. Shevade and C. Bhattacharyya and K.R.K. Murthy, “Improvements to platt’s smo algorithm for svm classifier design,” *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[42] Minka, Thomas P, “A comparison of numerical optimizers for logistic regression,” *Unpublished paper available at http://research.microsoft.com/en-us/um/people/minka/papers/logreg/minka-logreg.pdf*, March 2007.

[43] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[44] T. M. Tomita, M. Maggioni, and J. T. Vogelstein, “Randomer forests,” *arXiv preprint arXiv:1506.03410*, June 2015.

[45] D. Aha and D. Kibler, “Instance-based learning algorithms,” *Machine Learning*, vol. 6, pp. 37–66, 1991.

[46] Padraig Cunningham and Sarah Jane Delaney, “k-nearest neighbour classifiers,” University College Dublin, Tech. Rep. UCD-CSI-2007-4, March 2007. [Online]. Available: https://csiweb.ucd.ie/files/UCD-CSI-2007-4.pdf

[47] Mizutani, Eiji and Dreyfus, Stuart E, “On complexity analysis of supervised mlp-learning for algorithmic comparisons,” in *Neural Networks*, vol. 1. IEEE, 2001, pp. 347–352.

[48] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, “Automated topic naming supporting cross-project analysis of software maintenance activities,” *Journal of Empirical Software Engineering*, vol. 18(6), pp. 1125–1155, 2013. [Online]. Available: http://softwareprocess.ca/pubs/hindle2011EMSE-automated-topic-naming.pdf

[49] A. Hindle, N. Ernst, M. M. Godfrey, and J. Mylopoulos, “Automated topic naming to support cross-project analysis of software maintenance activities,” in *Proc. of 2011 Working Conference on Mining Software Repositories (MSR-11)*, 2011, inproceedings, pp. 163–172. [Online]. Available: http://softwareprocess.ca/pubs/hindle2011MSR-topicnaming.pdf

[50] Y. LeCun, C. Cortes, and C. J. Burges, “The mnist database of handwritten digits,” http://yann.lecun.

com/exdb/mnist/, 1998.
[51] "M. Lichman", “"UCI machine learning repository",” "2013". [Online]. Available: "http://archive.ics.uci.edu/ml"
[52] Machine Learning Laboratory, “Mnist arff files,” http://axon.cs.byu.edu/data/mnist/, 2015.
[53] M. Courbariaux, I. Hubara, C. D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-,” Feb 2016.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

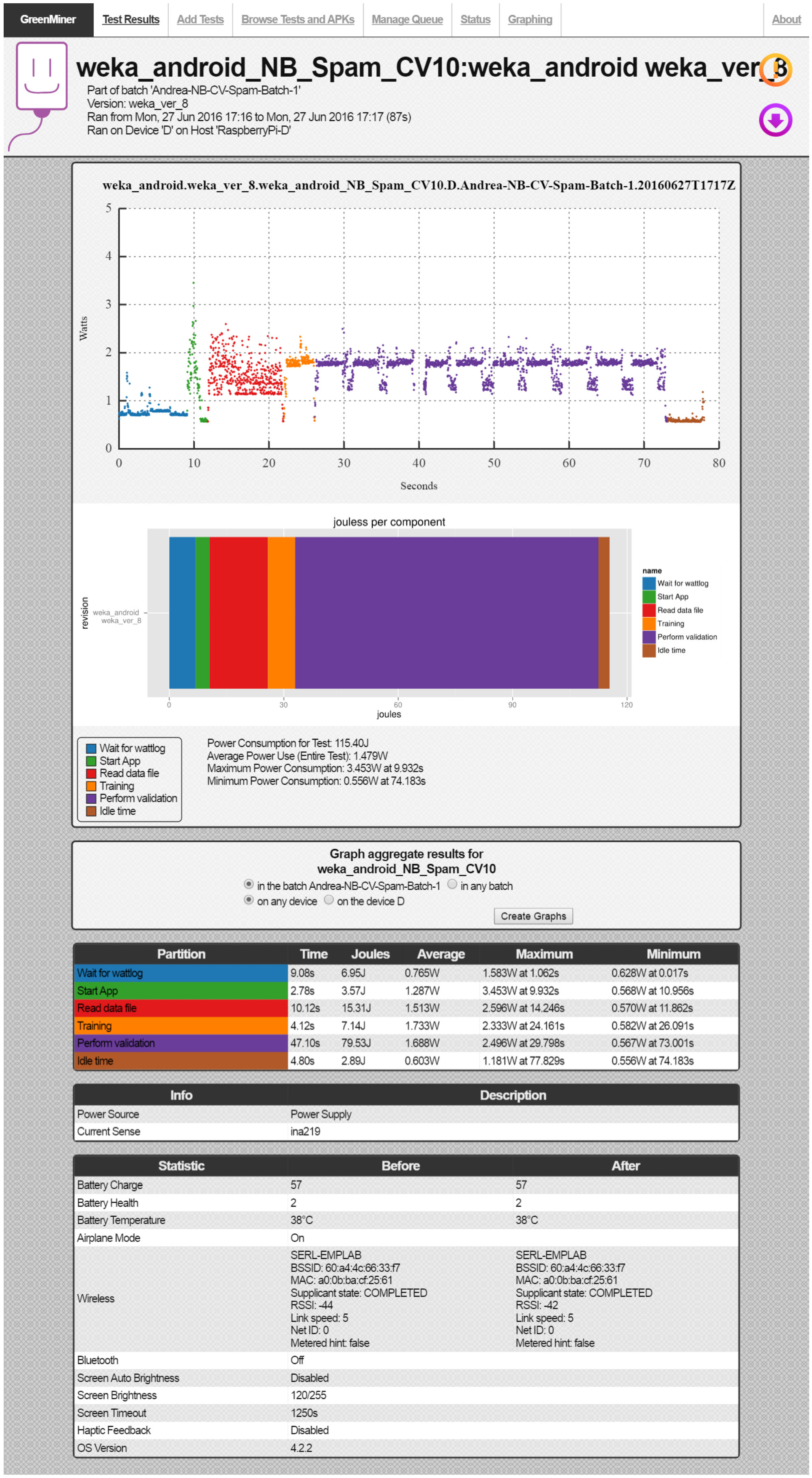

Figure 1. Example of a GreenMiner profile for a test run of 10-fold cross validation on Naïve Bayes with the Spambase dataset

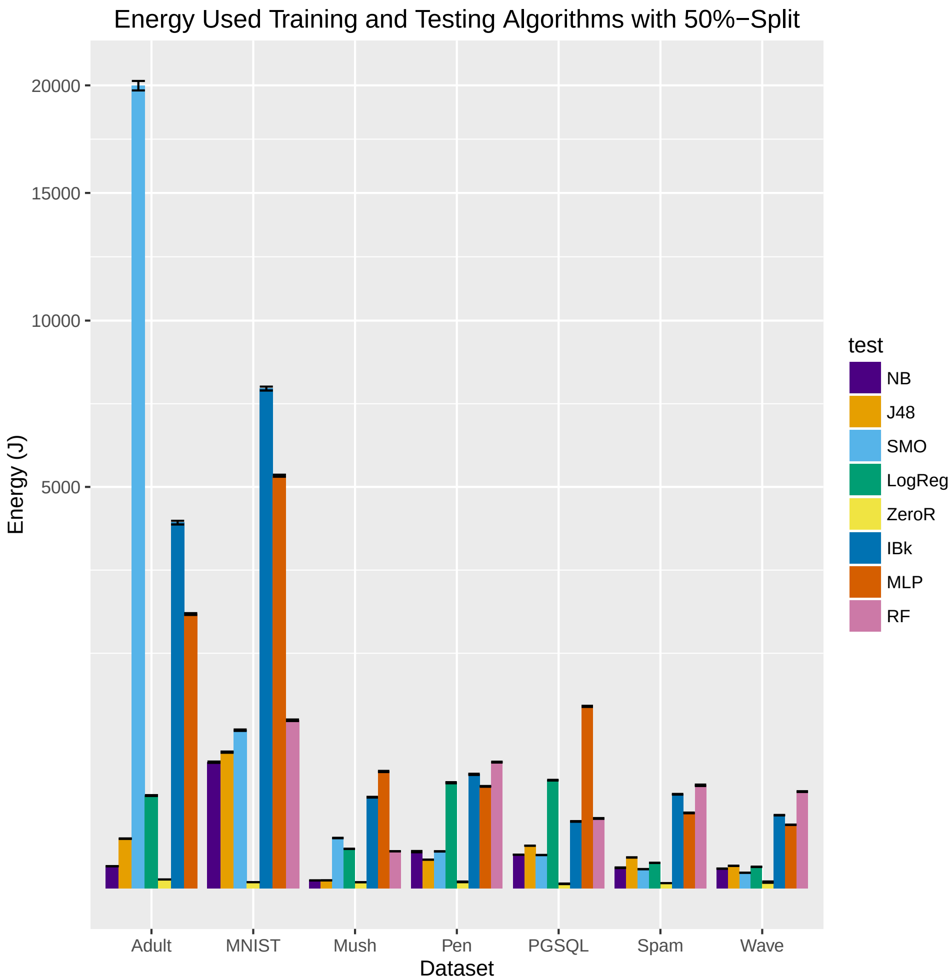

Figure 2. Energy consumption to train and test on 50% split

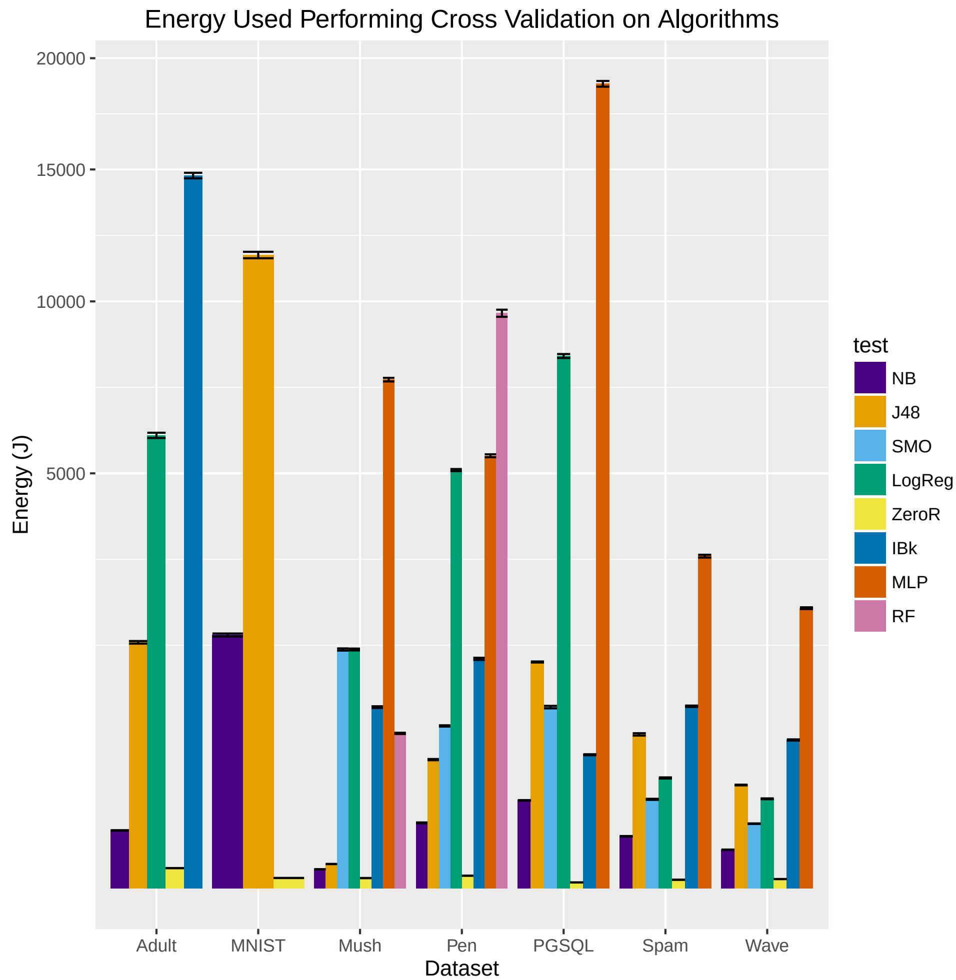

Figure 3. Energy consumption to perform 10-fold cross validation

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

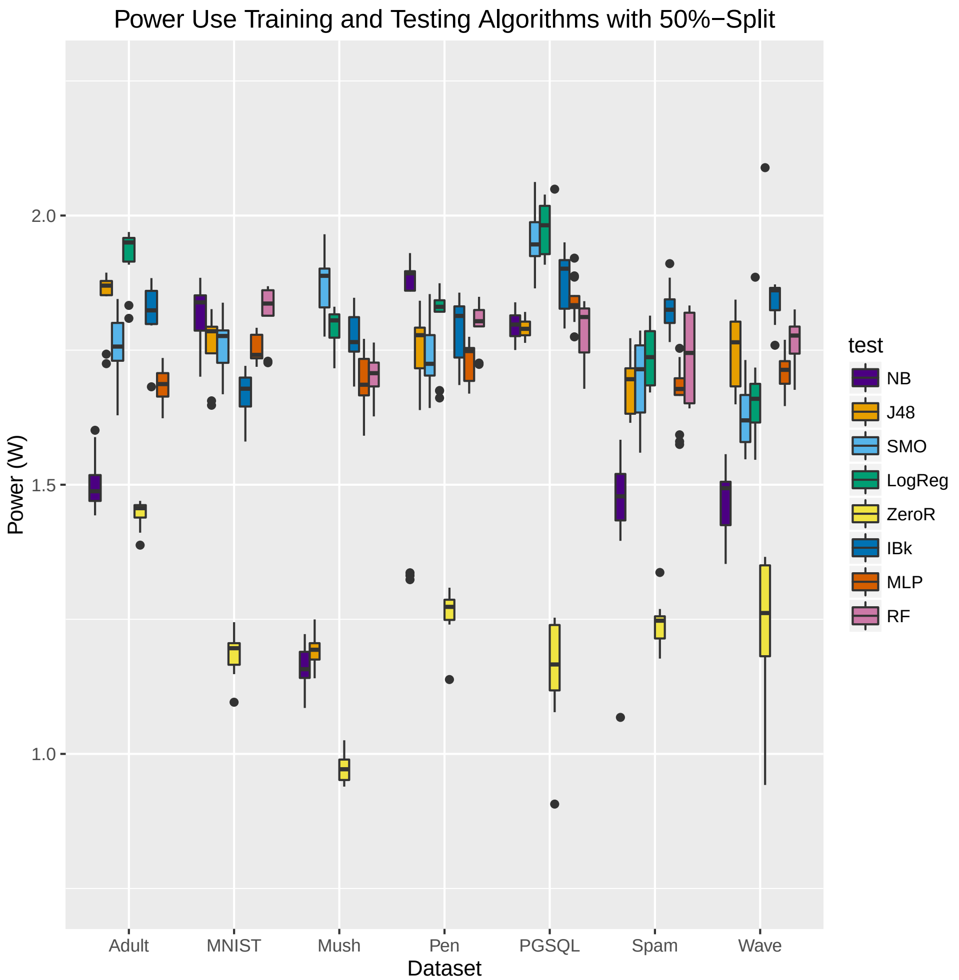

Figure 4. Power consumptionto train and test with 50% split

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

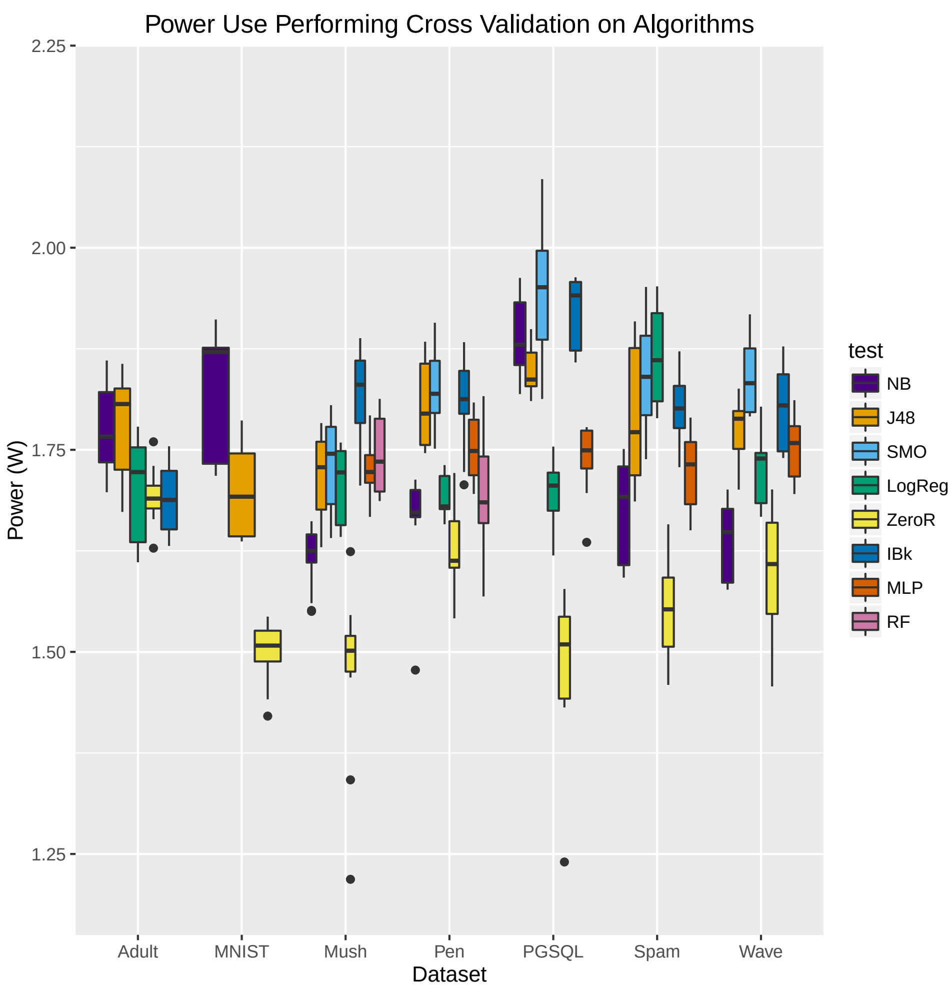

Figure 5. Power consumption to perform 10-fold cross validation

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

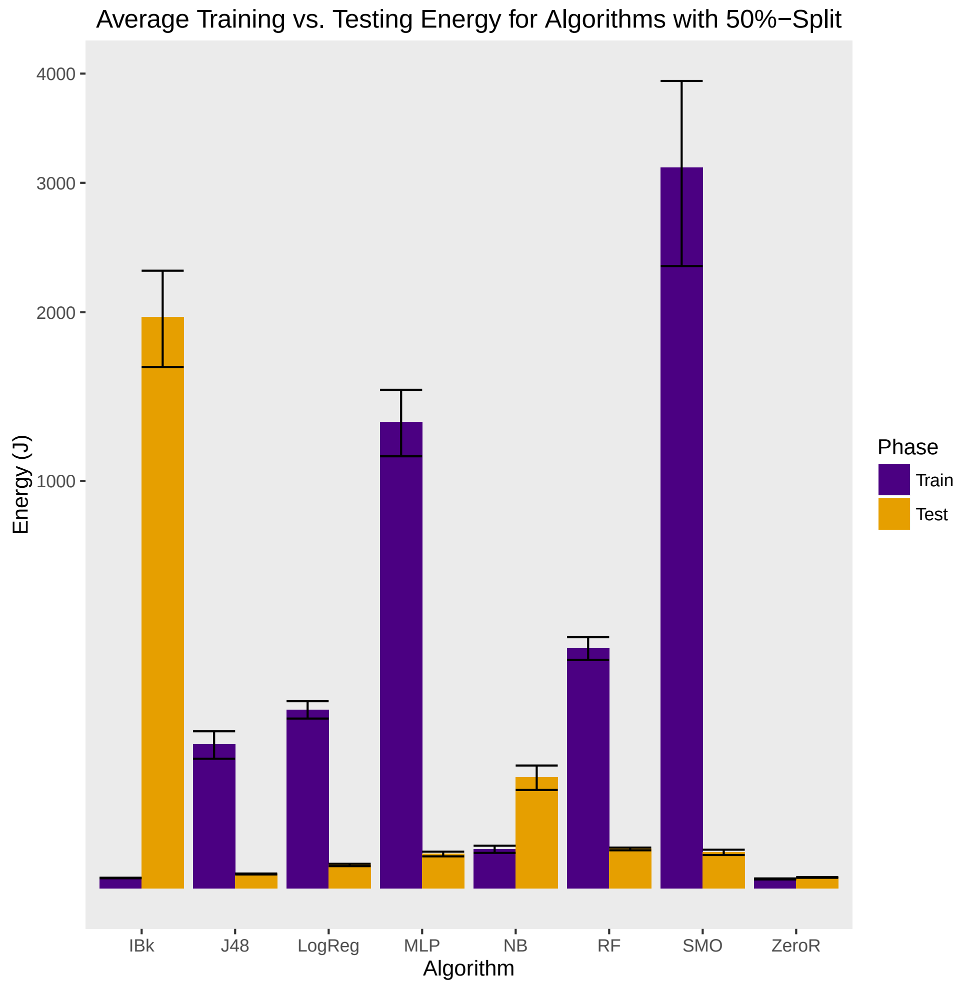

Figure 6. Comparison of average energy use training and testing algorithms with 50% split

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016

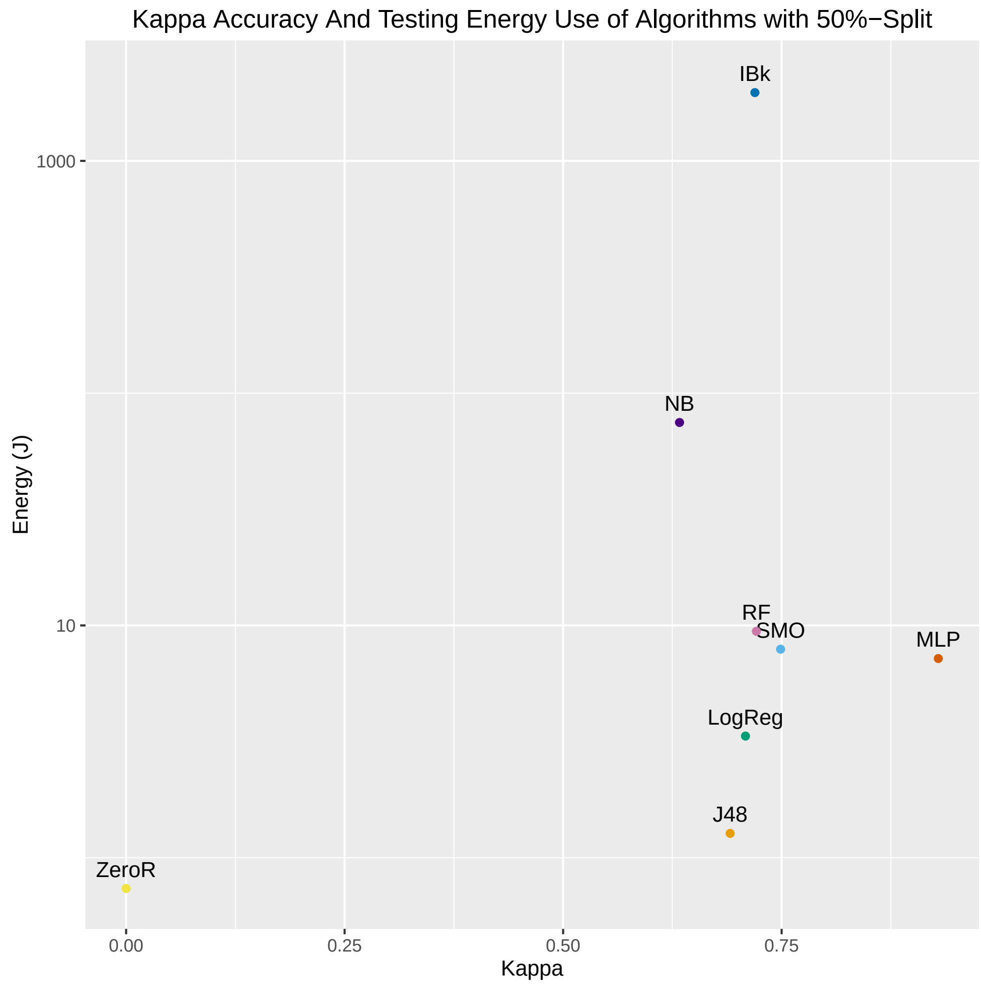

Figure 7. Scatterplot of energy consumption during classification (not training) versus Kappa.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.2431v1 | CC BY 4.0 Open Access | rec: 8 Sep 2016, publ: 8 Sep 2016