Predicting Battery Remaining Minutes based Related Features

Yijun Liu
University of California, San Diego
San Diego, California
Email: yil724@ucsd.edu

Kaixin Huang
University of California, San Diego
San Diego, California
Email: k3huang@ucsd.edu

Jinzong Que
University of California, San Diego
San Diego, California
Email: jque@ucsd.edu

1. Abstract

Our goal for this project is to understand and discover features that affect the battery's estimated remaining time. Through our exploratory data analysis, we have discovered eight features, namely the number of devices, number of processes, average memory, average page faults, designed capacity, cpu percentage, cpu seconds, and cpu temperature. Using these eight features, we decided to come up with several different models, Linear Regression, Decision Tree Regressor, SVM, Random Forest Regressor, Adaboost Regressor, Gradient Boosting Regressor and Bagging Regressor. To understand which model performs the best given these features, we performed hypothesis testing. In the end, our results show that Gradient Boosting Regressor performs the best out of all in that the maes generated on the train and test set are quite low and very similar. This indicates that Gradient Boosting Regressor has less of an overfitting issue than the other two models. Another indication is that through our hypothesis testing, our P-values indicate that Gradient Boosting Regressor performs the best among all others.

2. Introduction

Ever since the advent of personal computers and mobile devices, the attention has shifted to battery technology. Reasons for such are many, but one main factor is attributed to people's rise of expectation in the speed and longevity of these devices. In the early days, the slow and glitchy devices used to be the norm. In fact, having one was a luxury in and out of itself. However, as tech companies competed against each other to release the better product, these devices began to see a dramatic increase in their speed and longevity. This in turn led to groups of scientists and engineers coming together to investigate and delve deeper into the problem of lagging, and the solution they believed would exponentially increase people's device usage was to invest in battery technology. In an attempt to maintain the churn rate, groups of professionals flocked to battery experts to apply the latest technology onto their devices such that customer satisfaction would remain high. For such reasons, we would like to investigate the features that affect the battery's estimated remaining time.

Provided by the Intel Teams, we have decided to specifically focus on battery related datasets. Namely, some of these datasets would contain the needed GUIDs, or systems that we are interested in; the specified devices, which in our case, would be DC battery; cpu information, which provide insight to issues of overheating and cpu capabilities; process information, which allows us to understand the memory usage and page faults of these running processes. In the following sections, we will delve deeper into many of these features and eventually build a model based on the features we deem appropriate for our analysis.

3. Methods

3.1. Data Collection

This part emphasized on the accomplishments we have done for the last quarter: We practiced the Data Collection process – Our interested data fields for collecting are those we believed that would be related to the lifetime of batteries.

To retrieve data, we first replicated what was instructed on the main documentation [Int] from MSDN:

- Enumerate the battery devices through `SetupDiGetClassDevs` functiongetting the name and details of this battery through `SetupDiGetDeviceInterfaceDetail` and `SetupDiEnumDevice` Interfaces, and creating a handle to request information from it;
- To retrieve information, we first need to request for `Battery_Tag` through using `IOCTL_BATTERY_QUERY_TAG` control code, and with this tag, we are able to retrieve `Battery_Designed_Capacity`, `Battery_Full_Charged_Capacity`, and `Battery_Cycle_Count` through `IOCTL_BATTERY_QUERY_INFORMATION`

control code.

- To retrieve `Battery_Current_Capacity`, `Battery_Voltage`, `Battery_Rate`, we used control code `IOCTL_BATTERY_QUERY_STATUS`.

For other data we need, we also refer to `GetSystemPowerStatus` and `CallNtPowerInformation` functions. These two functions retrieve back `Battery_Left`, `Battery_Flag` , `Battery_Life_Time`, `Battery_Charging`, `Battery_Discharging`, `Battery_MaxCapacity`.

We revised our input library based on the sample provided on `static_standard_input`. Therefore, our input library is dependent on a static input library. The reason for using a static library is that the number and the type of our inputs data would not change over time.

With the self-designed input library, we run `ersv.exe` for about 2 hours with a time interval of 30 seconds to collect around 3000 records of data.

These practices on collecting data by building on our own input libraries gave us a better sense on how the drivers control the sensors on our computers for collecting multiple information.

3.2. Data Preparation

For the second quarter, unlike what we have done for the previous one, we were provided with the pre-collected data by the Teams. Therefore, all of our analysis are based on those datasets.

Our interested datasets are Battery Events related dataset (batt_acdc_events.csv000.gz), Battery Information related Dataset (batt_info.csv000.gz), Device usage related Dataset (devuse_4known_device.csv000.gz and devuse_4known_device.csv001.gz), CPU related Dataset (hw_metric_histo.csv000.gz and hw_metric_histo.csv001.gz), and Process related Dataset (plist_process_resource_util_13wks.csv000.gz).

Battery Events dataset would provide us with activities log for batteries, for example, it contains the information about whether the battery is running on Direct Current or Alternating Current; Battery Information dataset gave us a comprehension about the static information of batteries, for example, the full charged capacity for each battery; Device Dataset would enable us to understand what kind of device our system is currently running on; CPU related Dataset contains the information about CPU information, for example, current CPU temperature and CPU utilization.

Steps for collecting interested information are listed below:

- We started with Device Usage related Dataset with filtering conditions of i) device name must be ‘GUID_DEVICE_BATTERY’; and ii). The collected time should be within September of 2020. After loading Device Usage related Dataset, we obtained our interested GUIDs.
- In order to filter out only DC battery, we switched focus on Battery Events related Datasets. To filter out Battery Events dataset, we added the filtering conditions that the battery type should be ‘DC’ and the collected time should also be within September of 2020. After this, we get a new set of our needed GUIDs, and we explored other datasets based on those needed GUIDs.
- We utilized those GUIDs for further filtering on other datasets (e.g. Battery Information related Dataset, Process related Dataset and CPU related Dataset). For CPU related Datasets, one additional filtering condition we added is that the data must contain information about CPU utilization per each CPU core or information about CPU temperature in centigrade.

After collecting needed data, we manually selected 8 features for predicting Battery minutes remaining. From Battery Information related dataset, we selected Average Full Charge Capacity; From Process related dataset, we selected number of processes per guid, Average Page Faults per guid, Average Memory per guid, and Average CPU seconds per guid; From CPU related dataset, we selected Average CPU utilization per each CPU core and Average CPU temperature per guid; From Device Usage related dataset, we selected number of devices per guid.

3.3. EDA

We chose 8 features as our features on building our regression model for predicting battery remaining minutes. The reasons for selecting those 8 features are that 1). From our experiences, we realized that when we have multiple process or devices are going on, usually the battery would have a lower remaining time; 2). When memory related issue occurs, it would always affect the performance of batteries. Therefore, we considered Page Faults and Memory as another features to select; 3). Static information of battery, for example, the full charged capacity or designed capacity would definitely define the attribute of battery, making the performance of battery different; 4). CPU related information would also be a factor the influence the battery remaining time.

With our comprehension above, we did a correlation analysis between those selected 8 features and battery remaining minutes and the results are:

Target (y)	Features (Xi)	Correlation
battery minutes remaining	number of devices per guid	-0.009
	number of processes per guid	-0.023
	Average Page Faults per guid	-0.017
	Average Memory per guid	0.004
	Average CPU seconds per guid	-0.025
	Average Full Charge Capacity	-0.002
	Average CPU utilization per each CPU core	-0.004
	Average CPU temperature per guid	-0.005

Correlation Analysis indicates that there are weakly-negative correlation between battery minutes remaining and number of devices per guid, number of processes per guid, Average Page Faults per guid, Average CPU seconds per guid, Average Full Charge Capacity, Average CPU utilization per each CPU core, Average CPU temperature per guid. Even though the correlation values are pretty low, the negativity still confirms with our expectation as we believed that as the values of those features increase, battery minutes remaining should decrease. One possible explanation for the low value of correlation coefficients, as suggested by the Teams, is that those fields were mixed with both the DC and AC batteries, and we were unable to separate them.

3.4. Model

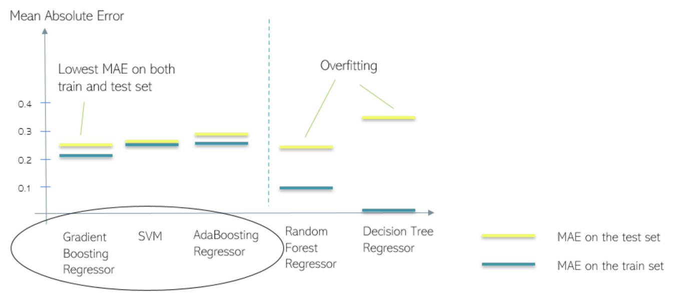
For the Model Part, we started our model building by using Linear Regression model, which is the most basic machine learning model to start, and later came up with some more complex models and finally selected 3 models with the lowest Mean Absolute Error. The reason for choosing Mean Absolute Error is that we are selecting a Regression Model, and we wants to see how varied our predictions are compared to the true values, no matter which directions those predictions are varied.

3.4.1. Baseline Model: Linear Regression

We started building our prediction model based on Linear Regression, the simplest regression model. After training on the model on the training dataset and test on the test dataset, we obtained a Mean Squared Error of 0.2656 on test dataset, which is lower than the mae of 0.2666 on the training set. We are not sure on the performance of this model, and want to see whether we could decrease mae further, so we switch to improved models.

3.4.2. Improved Model:

We tried different models and selected 3 models with the lowest maes as our improved models: Gradient Boosting Regressor, Support Vector Machine and AdaBoosting Regressor. For the other two models we have tried, we realized that these two models would overfit our data.



For our improved Model, we realized that only Gradient boosting Regressor, SVM, and Adaboosting Regressor do not occur the issue of over-fitting, so we are going to pay more attention on those and did Hypothesis Testing to see which one is the best improved model.

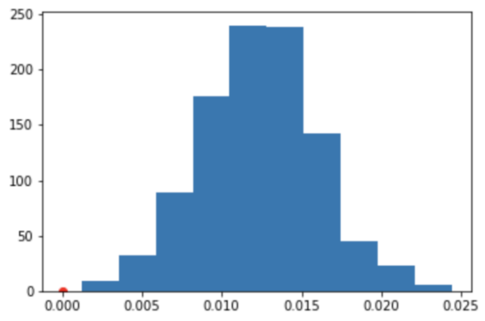
3.5. Hypothesis Testing

3.5.1. SVM vs Gradient Boosting Regressor:

In our hypothesis testing, we would like to understand the performance between the SVM and the Gradient Boosting Regressor on our Dataset.

- There's no difference in performance between SVM and Gradient Boosting Regressor.
- Gradient Boosting Regressor performs better than SVM.

Our test statistic for our hypothesis test is the observed difference between the MSE on the test set using SVM and the MSE on the test set using Gradient Boosting Regressor. With this, we then ran a simulation to generate new X and y by test, train, and split X and y in every new iteration, for a total of 1000 iterations. Our simulated differences in MSEs between the two models are displayed by the plot below:



Comparing our observed test statistic to our simulated test statistics, our p-value comes out to be 0.0. As a result, we reject the null hypothesis given our threshold of 0.05, but only by very slightly. This tells us that the Gradient Boosting Regressor does perform slightly better than SVM.

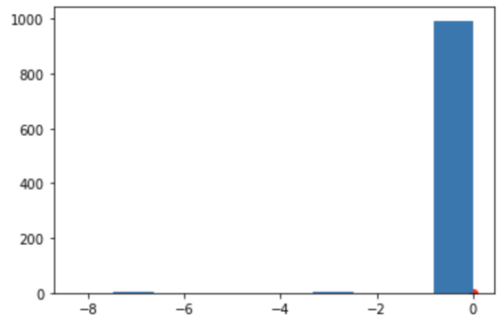
3.5.2. AdaBoosting Regressor vs Gradient Boosting Regression:

After comparing the performance between SVM and Gradient Boosting Regressor, we would like to understand the performance between AdaBoosting Regressor and Gradient Boosting Regressor on our Dataset. From above, we could see that the mae generated by the Gradient Boosting Regressor performed better in the sense that the mae on the test is lower than the mae on the test from AdaBoosting Regressor. Since we would like to verify if Gradient Boosting Regressor's performance is due to random chance or not, our null and alternative hypothesis are as follow:

- Null Hypothesis: There's no difference in performance between AdaBoosting Regressor and Gradient Boosting Regressor

- Alternative Hypothesis: Gradient Boosting Regressor performs better than AdaBoosting Regressor.

Our test statistic for our hypothesis test is the observed difference between the MSE on the test set using Gradient Boosting Regressor and the MSE on the test set using AdaBoosting Regressor. With this, we then ran a simulation to generate new X and y by test, train, and split X and y in every new iteration, for a total of 1000 iterations. Our simulated differences in MSEs between the two models are displayed by the plot below:



Comparing our observed test statistic to our simulated test statistics, our p-value comes out to be 0.015. As a result, we fail to reject the null hypothesis given our threshold of 0.05. This indicates that Gradient Boosting Regressor performs better than AdaBoosting Regressor.

3.6. Discussion and Limitations

After discussing our project with Teams, we realized that we have mainly three limitations:

- Features are not sufficient as we still need to consider the factors from users. For example, users playing games would always have a lower battery remaining time comparing to users using only basic operations;
- In evaluating battery remaining time, using average is not the best way as one situation might be that, the user initially do not plug in the charger, so the battery remaining time would be 180 minutes; but after he or she plugging in the charger, the remaining time would increase, say, to 240 minutes. Therefore, we anticipate the remaining minutes should be 240 minutes. But according to our averaging logic, we would get 210 minutes.
- For the feature of number of devices per guid, we ignored the factes that different devices would require different power of battery to run. Therefore, this might cause our correlation analysis be confounded;
- We noticed that the correlation coefficients are pretty low. One explanation is that our dataset is limited as fields such as CPU related information are mixed with both DC and AC data, which we were unable to separate.

3.7. Conclusion

Our goal is to understand the features that affect the battery's estimated remaining time. Through our exploratory data analysis, we have found eight features, namely the number of devices, number of processes, average memory, average page faults, designed capacity, cpu percentage, cpu seconds, and cpu temperature. With these features in hand, we then built several models. In the end, we discovered that out of all, Gradient Boosting Regressor performs the best.

References

[Int] IntelCo. *Enumerating Battery Devices*. URL: https://docs.microsoft.com/en-us/windows/win32/power/enumerating-battery-devices. (accessed: 05.31.2018).