ISE 4984/5984 Final Project

Tai-Jung Chen

Rechargeable Battery Degradation Prediction via Gradient Boosting Regression Tree and Extreme Gradient Boosting

1. Introduction

Nowadays, lithium-ion batteries have been widely used in our daily lives. To name a common-life application, lithium-ion batteries are serving the power systems for electric automobiles and mobile phones (**Fig. 1**). The reason for such applications is because its low cost, high energy density, and long lifetime properties. However, due to the sophisticated mechanism, the battery systems often entail delayed feedback performance. If one could accurately perform prediction using early cycle data, not only can the manufacturers optimize the production of the lithium-ion batteries, end-users could also benefit from the prediction to have a better user experience while using the devices which is powered by lithium-ion batteries.

Continuing the preliminary studies of the lifespan prediction of lithium-ion batteries from the mini project, this project aimed to decrease the prediction errors, which are Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). The MIT dataset which was generated by Severson, et al. [1] was studied and explored in this project.

Features related to voltage, current, and temperature of the charging process from the MIT dataset were extracted and served as inputs to the Gradient Boosting Regression Tree (GBRT) model and Extreme Gradient Boosting (XGBoost) model. 5-fold cross validation was conducted to find the best hyper-parameter such as learning rate, number of trees, and maximum number of splits of the tree for the GBRT and XGBoost. Python was used to implement the data preprocessing and model building. Some discussions of the model performances were also carried out. Last but not least, some ideas for the future outlook were brainstormed and given, hoping to further make contributions to the lifespan prediction of lithium-ion batteries research.

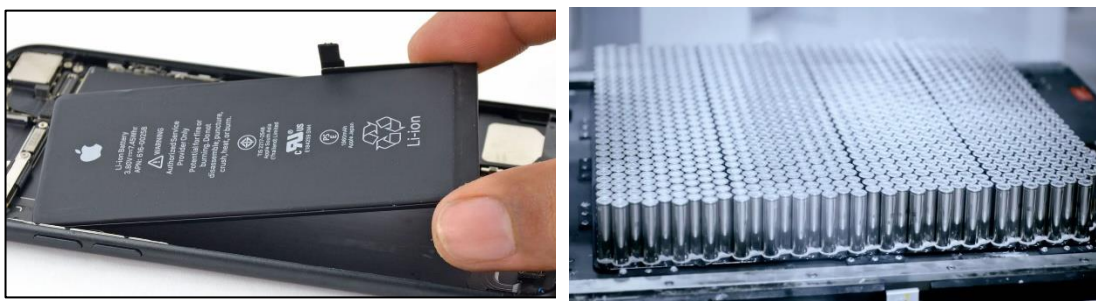


Fig. 1. The battery cell of Tesla

2. Data Preprocessing

2-1. Dataset

The dataset used in this project is the MIT dataset generated by Severson, et al. [1] The dataset consists of the charge-discharge cycle data from 124 commercial LFP/graphite cells (A123 Systems, model APR18650M1A, 1.1 Ah nominal capacity). The cells were cycled in a 48-channel Arbin LBT potentiostat under 30°C chamber temperature. To generate different cycle lifetimes, from approximately 150 to 2,300 cycles (average cycle life of 806 with a standard deviation of 377), the charging conditions varied in 72 different ways while the discharging condition remained the same (4 C to 2.0 V, where 1 C is 1.1 A) (**Fig. 2.**). The research team measured voltage, current, the cell can temperature and internal resistance during cycling. This information is critical and useful for us to construct and extract features from the dataset to perform advanced analysis.

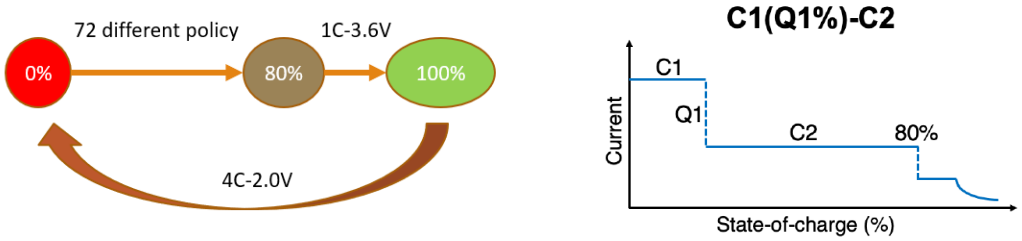


Fig. 2. Illustration of extreme fast charging policy structure

2-2. Feature Constructions

One of the valuable findings of the original research [1] from the MIT dataset discovered that the variance of the change in discharge voltage curves between different cycles was highly related to the degradation of lithium-ion batteries [1]. To be more specific, the original research denoted the change in discharge voltage curves between cycles i and j to be $\Delta Q_{j-i}(V) = Q_j(V) - Q_i(V)$, where $Q_i(V)$ *denotes the Discharge Capacity as a function of Voltage for the* i_{th} *cycle.* Furthermore, they calculated the statistics such as minimum, mean, and variance of $\Delta Q_{j-i}(V)$ of each cell, and found out that these statistics led to a clear trend between cycle life. For example, $\mathrm{VAR}(\Delta \mathrm{Q}_{100-10}(\mathrm{V}))$ showed a correlation of -0.92 with cycle life on a log-log axis. (**Fig. 3**)

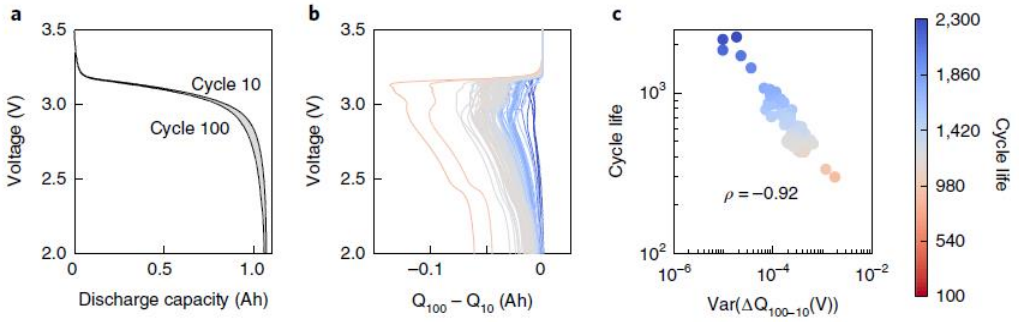


Fig. 3. $\Delta Q_{j-i}(V)$ is highly correlated to cycle life

In the light of the original research of the MIT dataset, Yang et al. [2] further defined features that were more representative. Yang et al. categorized the features into three categories, which were voltage-related features, current-related features, and temperature-related features (**Table 1**). Yang et al [2]. considered the differences between cycle 30 and cycle 100, cycle 30 and 150, cycle 30 and 200, and cycle 30 and 250. Therefore, each of the 18 features would be consisted of 4 sub-features, making the total number of features 72. These features were selected in this project as the reference for performing feature extraction.

To implement the ideas via Python, a Python package, pandas [3], was used to load the data from csv files to Python data frame. After loading the csv files to Python, numpy [4] package was used to turn the data into arrays and performed calculations between columns and rows of the arrays, and constructed features needed for model building. Cycle life extreme values, which were lying outside the $\bar{y} \pm 3\sigma_y$ region, were eliminated during the preprocessing (**Fig. 4-(a)** and **Fig. 4-(b)**). After the features were successfully extracted, a pickle file is made to serve as the input of the models in the next step. The Capacity-related features' parameter definition could be found as follows:

$$Model\ A\text{:}\ \ C_k = p_1 k + p_2$$

$$Model\ B\text{:}\ \ C_k = p_3\sqrt{k} + p_4$$

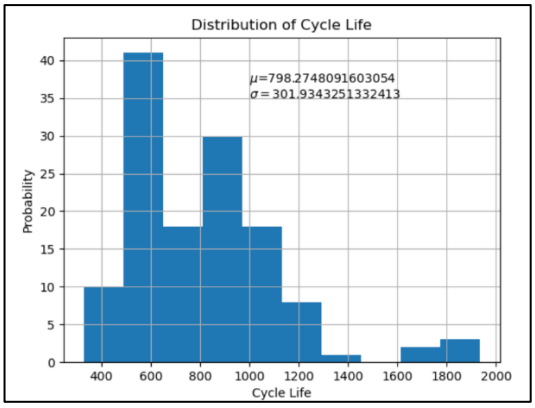
$$Model\ C\text{:}\ \ C_k = p_5 p_6^k + p_7$$

Where C_k is the discharge capacity at cycle k, p is the pre-determined model parameters. In domain knowledge of lithium-ion battery, p_6 in $Model\ C$ represents the coulumbic efficiency.

Table 1. The definition of the constructed features

(Note each feature could be turned into 4 sub-features) $I = \{30\}$; $J = \{100,150,200,250\}$

Type	Code and Definition
Voltage-related features	V1: Intensity decrease of IC peak from cycle *I* to *J* V2: Voltage shift of IC peak from cycle I to J V3: min{ $\Delta Q_{j-i}(V)$} V4: max{ $\Delta Q_{j-i}(V)$} V5: Mean{ $\Delta Q_{j-i}(V)$} V6: Var{ $\Delta Q_{j-i}(V)$} V7: Skewness{ $\Delta Q_{j-i}(V)$} V8: Kurtosis{ $\Delta Q_{j-i}(V)$}
Capacity-related features	C1: $p_1\ of\ Model\ A$ (fit to capacity fade curve from cycle *I* to *J*) C2: $p_2\ of\ Model\ A$ (fit to capacity fade curve from cycle *I* to *J*) C3: $p_3\ of\ Model\ B$ (fit to capacity fade curve from cycle *I* to *J*) C4: $p_4\ of\ Model\ B$ (fit to capacity fade curve from cycle *I* to *J*) C5: $p_5\ of\ Model\ C$ (fit to capacity fade curve from cycle *I* to *J*) C6: $p_6\ of\ Model\ C$ (fit to capacity fade curve from cycle *I* to *J*) C7: $p_7\ of\ Model\ C$ (fit to capacity fade curve from cycle *I* to *J*)
Temperature-related features	T1: average surface temperature of the discharge process, difference between cycle *I* to *J* T2: maximum surface temperature of the discharge process, difference between cycle *I* to *J* T3: minimum surface temperature of the discharge process, difference between cycle *I* to *J*



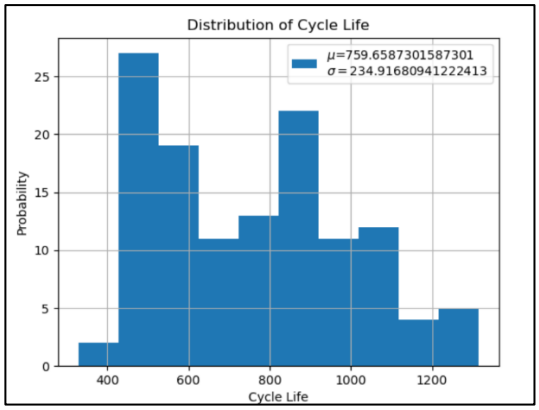


Fig. 4-(a). The histogram of cycle lives before eliminating extreme values

Fig. 4-(b). The histogram of cycle lives after eliminating extreme values

3. Modeling

3-1. GBRT model

The GBRT model is a powerful ensemble model. By aggregating lots of weak decision trees via Boosting algorithm, GBRT can have a good performance on predictions [2]. Since predicting the lifespan of lithium-ion batteries is a regression problem, the main objective for GBRT will be reducing the residuals. Thus, the squared error formula was picked as the loss function (**eq. (1)**). To prevent overfitting and increase the model's generalization ability, a parameter called learning rate (ν in **eq. (2)**) is often used in gradient boosting. In addition to learning rate, one can tune the number of splits per weak decision tree, which is the J_m in **Table 2** - step 2. Last but not least, one can also set the number of weak decision trees, which is the M in **Table 2** - step 2. These hyper-parameters are often used to prevent overfitting during the training process so that the model can be generalized enough to perform good predictions on the testing phase.

$$L\big(y, F_M(x)\big) = \sum_{i=1}^{N}\big(y_i - F_M(x_i)\big)^2 \quad \textbf{(1)}$$

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}) \quad \textbf{(2)}$$

where y_i is the true value of the i_{th} data, and $F_M(x_i)$ is the predicted value of the i_{th} data.

Table 2. GBRT algorithm

Input: $Data\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x_i))$

Step 1: initialize model with a constant value: $F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for m=1 to M:

(A) Compute $\gamma_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad for\ i = 1, \dots, n$

(B) Fit a regression tree to the γ_{im} values and create terminal regions $R_{jm}, for\ j = 1, \dots, J_m$

(C) $For\ j = 1, \dots, J_m$, compute $\gamma_{jm} = argmin_\gamma \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i + \gamma))$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Step 3: Output $F_M(x)$

3-2. XGBoost model

Yang et al.[2] had shown that using GBRT could effectively reduce the prediction error on the MIT dataset. Sparked from this finding, XGBoost[5] model which is an enhanced version of the GBRT model was implemented in this project. XGBoost has been widely recognized by the machine learning community. Take the popular challenges on the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles [6].

XGBoost has some slight differences with GBRT. First, XGBoost's objective function takes regularization terms $\Omega(f_k)$ into account (**eq. (3)**). $\Omega(f_k)$ can be further decomposed into complexity and L2 norm regularization (**eq. (4)**). In **eq. (4)**, γ is a hyper-parameter that can be tuned to penalize large tree complexity. On the other hand, λ is another hyper-parameter that controls the ridge regularization.

$$\text{Objective: } L(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \quad f_k \in \mathcal{F} \qquad \textbf{(3)}$$

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda W^2 \qquad \textbf{(4)}$$

Where f_k is a decision tree; $\mathcal{F}\ is\ the\ space\ of\ all\ decision\ tree$;
$T\ is\ the\ number\ of\ nodes\ of\ the\ tree; W\ is\ the\ leaf\ weight\ of\ the\ tree$

Another difference between GBRT and XGBoost is that XGBoost utilizes the information from the second-order derivative of the objective function. To be more specific, the objective function is approximated using the second-order approximation (**eq. (5)**). Finally, we get the final form of the objective function for XGBoost in **eq. (6)**. Learning rate η is also applicable in the XGBoost model in **eq. (7)**. We hope to reduce

the prediction error by tuning the λ and γ while remain the ensemble structure which was observed to perform well in the prediction of lithium-ion battery lifespan.

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[L\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) \quad \textbf{(5)}$$

Where $g_i = \frac{\partial L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}} = \frac{\partial\left(\hat{y}_i^{(t-1)} - y_i\right)^2}{\partial \hat{y}_i^{(t-1)}} = 2\left(\hat{y}_i^{(t-1)} - y_i\right);$

$$h_i = \frac{\partial^2 L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial(\hat{y}_i^{(t-1)})^2} = \frac{\partial\left(\hat{y}_i^{(t-1)} - y_i\right)^2}{\partial(\hat{y}_i^{(t-1)})^2} = 2$$

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T \quad \textbf{(6)}$$

Where $G_j = \sum_{i\in I_j} g_i$; $H_j = \sum_{i\in I_j} h_i$

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta f_t(x) \quad \textbf{(7)}$$

Where $\hat{y}^{(t)}$ is the predicted value at time t

3-3. Splitting of Training Set, Validation Set, and Testing Set

The data frame processed in the previous section consisted of 126 examples and 72 features. A randomly train test splitting was performed by the ratio of $\frac{2}{3}$ for training and validation set, and $\frac{1}{3}$ for testing set. The training validation set was further split into validation set and training set in a 5-fold cross validation fashion to search for the optimal hyper-parameter combination.

3-4. Python packages for modeling

Scikit-learn package [7] was used to implement the model building and training process. It is a powerful package that could do train test splitting, model fitting, k-fold cross validation, and model score evaluation. On the other hand, XGBoost has its own package called “xgboost”. Although one can use Scikit-learn package to implement XGBoost model, xgboost package can compute faster than scikit-learn package. Therefore, we used xgboost to implement the XGBoost model.

4. Results

4-1. Metrics

Since the models used in this project are all regression models, the appropriate metrics would be Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), R square (R^2), and $Adjusted$ R square (R^2_{adj}) (**eq. (8) – (12)**). The formula for the above metrics could be found below:

$$RMSE = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2} \quad \textbf{(8)}$$

$$MAE = \frac{1}{n}\sum|y_i - \hat{y}_i| \quad \textbf{(9)}$$

$$MAPE = \frac{1}{n}\frac{\sum|y_i - \hat{y}_i|}{y_i} * 100\% \quad \textbf{(10)}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} = 1 - \frac{SSE}{SST} \quad \textbf{(11)}$$

$$R^2_{adj} = 1 - \frac{(1-R^2)(n-1)}{n-p-1} \quad \textbf{(12)}$$

Note that n denotes the number of battery samples; y_i and $\hat{y}_i$ represents the true values and the predicted values for sample i; p is the number of features used to build the model. The MAE measures how close estimates are to the corresponding outcomes. The $RMSE$, which characterizes the variation in errors, is more sensitive to large errors than the MAE. The $MAPE$ expresses the error as a percentage and evaluates the performance in terms of relative error. In these three metrics, small values indicate good model performance and high values indicate poor model performance. The R^2 is a percentage-based metric. It indicates how well the model is fitting the data. Ideally the R^2 should be as close as possible to 100% or 1. While the R^2 has a flaw. The value of R^2 could be increased by simply adding more features (since the SSE would decrease in that case). Therefore, R^2_{adj} is used to penalize useless features.

4-2. Optimal Hyper-parameters

4-2-1. hyper-parameter tuning for GBRT

The training set and validation set split in 3-3 were fitted to the GBRT model. In this project, learning rate, number of trees, and maximum number of splits were selected as the hyper-parameters to fine-tune the GBRT model.

In order to search for the best matches of hyper parameter, which were learning rate, number of trees, and maximum number of splits, a 5-fold cross validation was performed. A triple for loop is implemented in Python. Three lists of hyper parameters, which were learning rate (0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5), number of trees (100, 200, 300, 400, 500, 600, 700, 800, 900, 1000), and maximum number of split (1, 2, 4, 8, 16, 32, 64), were looped over by the triple for loop to run through all the combinations of the three hyper parameters. $RMSE$ was chosen as the criterion of the hyper parameter selection. The best hyper parameters obtained by the cross validation were: learning rate = 0.05, number of trees = 300, and maximum number of split = 1, with $RMSE = 80.1449$, $MAE = 59.2318$, $MAPE =$

7.2907%, $R^2 = 0.8846$, and $R^2_{adj} = 0.7278$. An illustration of how the cross validation error looked like under certain hyper-parameter combinations is presented below (**Fig. 5** and **Table 3**).

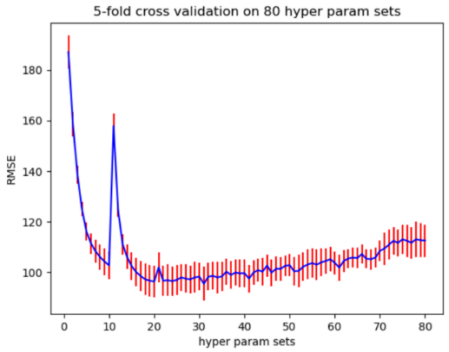


Fig. 5. The error bar chart of the cross validation RMSE for the 80 combinations of hyper parameter sets (fixing maximum number of splits to 1)

Table 3. cross validation results for GBRT

Hyper parameter sets	RMSE	MAE	MAPE	R^2	R^2_{adj}
best_lr: 0.25, best_num_tree: 900, best_max_split: 8	113.41	88.23	13.53%	0.7661	0.4483
best_lr: 0.2, best_num_tree: 700, best_max_split: 1	119.21	96.17	12.37%	0.7355	0.3762
best_lr: 0.15, best_num_tree: 300, best_max_split: 4	100.01	77.61	10.64%	0.8292	0.5972
best_lr: 0.2, best_num_tree: 800, best_max_split: 32	98.03	75.26	10.49%	0.8208	0.5774
best_lr: 0.5, best_num_tree: 1000, best_max_split: 8	114.65	83.25	11.62%	0.7158	0.3297
best_lr: 0.1, best_num_tree: 600, best_max_split: 4	88.81	70.99	10.17%	0.836	0.6132
best_lr: 0.25, best_num_tree: 1000, best_max_split: 32	84.96	61.33	8.02%	0.8781	0.7125
best_lr: 0.05, best_num_tree: 500, best_max_split: 4	104.40	81.81	10.63%	0.8085	0.5483
best_lr: 0.25, best_num_tree: 700, best_max_split: 8	96.35	73.23	9.49%	0.8496	0.6453
best_lr: 0.15, best_num_tree: 900, best_max_split: 4	114.19	89.24	13.44%	0.7567	0.4262
best_lr: 0.01, best_num_tree: 1000, best_max_split: 1	116.05	90.87	12.65%	0.7708	0.4594
best_lr: 0.2, best_num_tree: 100, best_max_split: 2	122.24	92.52	14.33%	0.7363	0.3781
best_lr: 0.15, best_num_tree: 500, best_max_split: 4	107.16	77.11	11.53%	0.7857	0.4946
best_lr: 0.05, best_num_tree: 400, best_max_split: 2	95.55	68.28	9.96%	0.8483	0.6422
best_lr: 0.15, best_num_tree: 100, best_max_split: 4	100.24	76.54	9.58%	0.8096	0.5509
best_lr: 0.01, best_num_tree: 600, best_max_split: 2	91.26	69.02	8.59%	0.8552	0.6585
best_lr: 0.5, best_num_tree: 800, best_max_split: 1	136.90	98.75	11.11%	0.759	0.4316
best_lr: 0.05, best_num_tree: 300, best_max_split: 1	80.14	59.23	7.29%	0.8846	0.7278
best_lr: 0.5, best_num_tree: 900, best_max_split: 32	113.99	87.18	11.86%	0.775	0.4693
best_lr: 0.25, best_num_tree: 600, best_max_split: 4	125.27	97.37	11.81%	0.7664	0.4491
Mean	106.14	80.70	10.96%	0.7994	0.5268
Standard Error	3.28	2.62	0.41	0.01	0.02

4-2-2 hyper-parameter tuning for XGBoost

Similarly, for the XGBoost model, learning rate, number of trees, maximum number of splits, lambda, and gamma were selected as the hyper-parameters. Like the above procedure, 5-fold cross-validation was performed on each combination of these hyper-parameter. To make a robust comparison with the GBRT model, I set the learning rate, number of trees, and maximum number of split to the same value as the GBRT model, then I added two lists which are lambda (1-100), and gamma (1-100). In this fashion, we can observe whether the regularization (represented by lambda) and pruning (represented by gamma) would reduce the prediction error.

The same procedure for selecting the best hyper-parameters mentioned in 4-2-1 was adopted (**Fig. 6**). 10 epochs of training and testing were executed. Each epoch represented a random training set and testing set splitting. The best hyper-parameter of each epoch as well as the metrics ($RMSE, MAE, MAPE, R^2, R^2_{adj}$) were shown in **Table 4**.

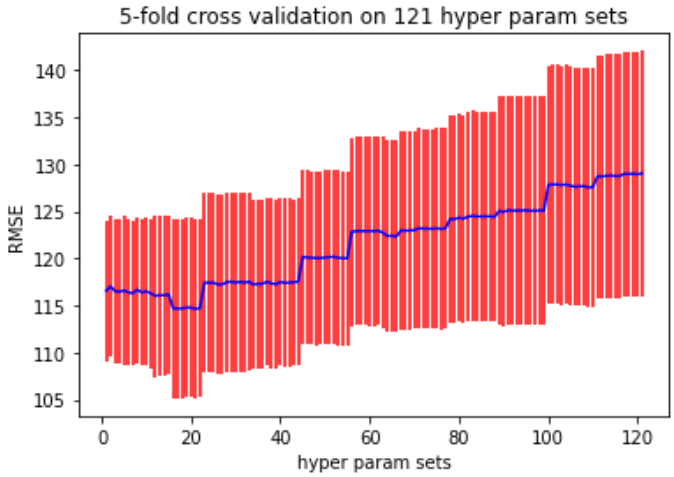


Fig. 6. RMSE of different hyper-parameter sets

Table 4. cross validation results for XGBoost

Hyper parameter sets	RMSE	MAE	MAPE	R^2	R^2_{adj}
best_lambda: 20 , best_gamma:30	96.01	61.86	7.58%	0.8490	0.6439
best_lambda: 0.0, best_gamma: 30.0	121.12	88.55	12.31%	0.7226	0.3457
best_lambda: 0.0, best_gamma: 20.0	103.57	83.05	10.84%	0.7751	0.4695
best_lambda: 0.0, best_gamma: 90.0	130.90	94.46	12.36%	0.7687	0.4544
best_lambda: 0.0, best_gamma: 20.0	122.85	83.02	10.05%	0.7746	0.4685
best_lambda: 0.0, best_gamma: 90.0	95.83	71.50	9.63%	0.8538	0.6553
best_lambda: 0.0, best_gamma: 40.0	87.20	61.86	7.84%	0.8692	0.6916
best_lambda: 0.0, best_gamma: 70.0	93.34	71.46	10.03%	0.8036	0.5367
best_lambda: 10.0, best_gamma: 70.0	96.60	63.97	8.15%	0.8344	0.6093
best_lambda: 0.0, best_gamma: 80.0	92.96	69.82	9.90%	0.8094	0.5505
best_lambda: 0.0, best_gamma: 90.0	101.59	78.92	10.25%	0.7619	0.4384
best_lambda: 0.0, best_gamma: 0.0	102.94	80.33	10.21%	0.7419	0.3913
best_lambda: 0.0, best_gamma: 60.0	99.53	82.12	11.22%	0.8247	0.5866
best_lambda: 0.0, best_gamma: 90.0	82.58	59.11	8.40%	0.8367	0.6148
best_lambda: 0.0, best_gamma: 40.0	99.42	74.24	9.77%	0.7827	0.4875
best_lambda: 0.0, best_gamma: 80.0	102.00	76.45	9.92%	0.7646	0.4448
best_lambda: 20.0, best_gamma: 60.0	108.60	84.42	11.68%	0.8037	0.5370
best_lambda: 0.0, best_gamma: 40.0	98.85	75.63	9.70%	0.7755	0.4705
best_lambda: 0.0, best_gamma: 60.0	97.77	77.76	10.07%	0.7856	0.4943
best_lambda: 0.0, best_gamma: 100.0	94.08	73.55	9.50%	0.7965	0.5200
Mean	**101.39**	**75.60**	**10%**	**0.8000**	**0.5200**
Standard Error	**2.63**	**2.08**	**0.0029**	**0.0087**	**0.0205**

4-3. Testing results

4-3-1 Testing on GBRT

After selecting the best hyper-parameters, we select learning rate = 0.05, number of trees = 300, and maximum split = 1 as the best hyper-parameter for GBRT model. we ran the GBRT model associated with the selected hyper-parameter on 10 different training and testing set to evaluate how generalized it was. The results were summarized in **Table 5**.

Table 5. Testing results on GBRT					
GBRT	RMSE	MAE	MAPE	R^2	R^2_{adj}
lr = 0.05; num_tree=300; max_split=1	123.76	92.66	11.58%	0.7714	0.4608
	117.06	84.63	10.27%	0.7635	0.4423
	88.30	71.25	8.70%	0.8757	0.7068
	95.08	69.86	8.29%	0.8313	0.6022
	89.34	67.93	9.53%	0.8357	0.6125
	84.59	62.42	8.05%	0.8404	0.6235
	103.96	77.36	10.79%	0.8500	0.6463
	103.53	74.74	9.31%	0.8254	0.5883
	116.59	90.17	12.47%	0.7194	0.3381
	111.83	87.18	13.26%	0.7598	0.4335
Mean	**103.40**	**77.82**	**10.22%**	**0.8073**	**0.5454**
Standard Error	**4.34**	**3.26**	**0.0056**	**0.0158**	**0.0373**

4-3-2 Testing on XGBoost

Similarly, we selected learning rate = 0.05, number of trees = 300, maximum split = 1, lambda = 1, and gamma = 90 as the best hyper-parameters for XGBoost model. We ran 10 different training and testing set to evaluate how generalized the XGBoost model was. The results were summarized in **Table 6**.

Table 6. Testing results on XGBoost					
XGBoost	RMSE	MAE	MAPE	R^2	R^2_{adj}
lr = 0.05; num_tree=300; max_split=1; lmb = 0; gamma = 90	121.80	89.60	11.20%	0.7785	0.4777
	117.07	88.3436	10.80%	0.7635	0.4421
	84.59	62.5076	7.66%	0.8859	0.7309
	86.38	63.9499	7.78%	0.8608	0.6716
	101.93	76.4085	10.68%	0.7861	0.4955
	89.73	65.7143	8.87%	0.8204	0.5763
	112.40	84.7777	12.45%	0.8247	0.5866
	97.86	75.5245	10.13%	0.8440	0.6321
	115.56	90.2687	12.85%	0.7243	0.3498
	102.38	79.846	11.84%	0.7987	0.5252
Mean	**102.97**	**77.69**	**10.43%**	**0.8087**	**0.5488**
Standard Error	**4.24**	**3.40**	**0.0058**	**0.0153**	**0.0360**

4-4. Comparison

To compare the results of GBRT model and XGBoost model, I performed testing on both models with the same training and testing set. The results were summarized

in **Table 7**. The average RMSE of 10 different epochs decrease 0.431 after using XGBoost. The standard error of this average RMSE was also smaller than the one using GBRT. Additionally, both models could achieve the RMSE around 84. The original MIT paper's "full" model, which is really selecting the features related to variance of $\Delta Q(V)$ and some discharge-related features (see **Appendix 1** for details), had a 100 RMSE. Since the data splitting strategy and feature construction of the original MIT paper was different from this final project, comparing with the model in the MIT paper might not be robust, but still showed some differences between these three models.

Table 7. Comparison between GBRT and XGBoost

GBRT performance	**RMSE**	**MAE**	**MAPE**	R^2	R^2_{adj}
Mean	103.40	77.82	10.22%	0.8073	0.5454
Standard Error	4.34	3.26	0.0056	0.0158	0.0373
XGBoost performance	**RMSE**	**MAE**	**MAPE**	R^2	R^2_{adj}
Mean	102.97	77.69	10.43%	0.8087	0.5488
Standard Error	4.24	3.40	0.0058	0.0153	0.0360

5. Conclusion

In this project, I applied the XGBoost model on the MIT dataset [1] to achieve a lower RMSE on the prediction of the lithium-ion battery lifespan. 72 features related to voltage, current, and temperature were constructed. A comparison with the GBRT model [2] was carried out. It was shown that XGBoost could slightly improve the prediction and was more convergent since it has a smaller standard error comparing to GBRT model.

This project has a large dataset to preprocess, a bit more sophisticated domain knowledge on the subject, and a more rigorous training process. I learned how to extract features in a logical way, how to perform cross validation techniques and find the best hyper-parameters in an appropriate way, and how to compare the results between different models robustly.

6. Future outlook

I would like to further tune the hyper-parameters on XGBoost model, since the hyper-parameter selected here were fixing the learning rate, number of trees, and number of splits same as the GBRT model. XGBoost also has other features that I could applied to improve the prediction such as early stopping and column subsampling.

In terms of feature extraction of the dataset, Xu et al. [8] applied a deep learning-based stacked denoising autoencoder (SDAE) method with a clustering by fast search (CFS) method to further select the essential features from the dataset. This could be adopted in the future. Additionally, the cycle number to choose from is also an

interesting question to dive into in future studies.

Classification is another aspect to discover because this could help manufacturers to better allocated their resources and make better decisions based on the classification results. The original research paper of the MIT dataset [1] simply utilized the data from the first 5 cycles to classify the batteries into two groups (thresholding at 550 cycles), which is the high-lifetime group and low-lifetime group. I built a simple classification model via XGBoost since it could also be used to classify data. Yet, some sampling and hyper-parameter tuning tasks still have to be done. I would do more digging on the classification problem of the lithium-ion battery in the future.

References

[1] K.A. Severson, P.M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M.H. Chen, M. Aykol, P.K. Herring, D. Fraggedakis, Data-driven prediction of battery cycle life before capacity degradation, Nat. Energy 4 (5) (2019) 383.

[2] F. Yang, D. Wang, F. Xu, Z. Huang, K. Tsui, Lifespan prediction of lithium-ion batteries based on various extracted features and gradient boosting regression tree model, Journal of Power Sources 476 (2020) 228654.

[3] McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

[4] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2.

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM.

[6] M. Johnen, S. Pitzen, U. Kamps, M. Kateri, P. Dechent, D. U. Sauer, Modeling long-term capacity degradation of lithium-ion batteries, Journal of Energy Storage 34 (2021) 102011

[7] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[8] F. Xu, F. Yang, Z. Fei, Z. Huang, K. Tsui, Life prediction of lithium-ion batteries based on stacked denoising autoencoders, Reliability Engineering and System Safety 208 (2021) 107396

Data Availability

The MIT dataset could be accessed at the following link:
https://data.matr.io/1/projects/5c48dd2bc625d700019f3204

Appendix 1.

	Features	"Variance"	"Discharge"	"Full"
$\Delta Q_{100\text{-}10}(V)$ features	Minimum		V	V
	Mean			
	Variance	V	V	V
	Skewness		V	
	Kurtosis		V	
	Value at 2V			
Discharge capacity fade curve features	Slope of the linear fit to the capacity fade curve, cycles 2 to 100			V
	Intercept of the linear fit to capacity fade curve, cycles 2 to 100			V
	Slope of the linear fit to the capacity fade curve, cycles 91 to 100			
	Intercept of the linear fit to capacity fade curve, cycles 91 to 100			
	Discharge capacity, cycle 2		V	V
	Difference between max discharge capacity and cycle 2		V	
	Discharge capacity, cycle 100			
Other features	Average charge time, first 5 cycles			V
	Maximum temperature, cycles 2 to 100			
	Minimum temperature, cycles 2 to 100			
	Integral of temperature over time, cycles 2 to 100			V

	Internal resistance, cycle 2			
	Minimum internal resistance, cycles 2 to 100			V
	Internal resistance, difference between cycle 100 and cycle 2			V