The Project Report for

# "GBatteries: Battery Health and Charge Speed Prediction"

Submitted in partial fulfilment for a course

MIE1628 - Big Data Science

Submitted by


| | |
|---|---|
| Vijaykumar Maraviya | (1006040320) |
| Ibrahim Shodeko | (1005664835) |
| Yerbolat Muldagaliyev | (1006449906) |


Guided by


| | |
|---|---|
| Dr. Yanina Shevchenko | Professor, University of Toronto |
| Seran Thirugnanam | GBatteries Inc. |
| Patrick Watt | GBatteries Inc. |




Department of Mechanical and Industrial Engineering

University of Toronto

# Contents

# 1 Introduction

Charging speed is one of the most significant factors for batteries. To rival the gasoline-powered vehicles, and hence reduce transportation-related emissions, electric vehicles need batteries that charge quickly. Moreover, enabling truly fast charging will improve the experience of billions of battery-powered devices such as cellphones, computers, power tools, drones, robots, and so on. Though the caveat is that the fast charging reduces the longevity of a battery. The faster the charging, the shorter the life of a battery.

## 1.1 GBatteries approach and challenges

GBatteries Inc. is developing protocols for fast-charging of off-the-shelf li-ion batteries that defy the relationship between speed and longevity. A protocol generates the complex pulses for charging based on values of 36 charging parameters. The values of these parameters can be adapted within a charging cycle or between cycles based on real-time monitoring and analysis of a battery's internal state. The GBatteries Active Battery Management System (ABMS) periodically measures a degradation indicator (di) while charging. Higher di values are associated with more degradation of a cell in the long term. The values of charging parameters can be adapted to reduce di. However, the drawback to adjusting parameters in this way is that it usually also reduces the charging speed. But It is observed that, for some magical set of parameter values, the inverse relationship between longevity and the speed is defied.

The goal when developing the protocol is to find the best parameters that minimize the degradation and meet or exceed the charging speed requirement while complying with the system constraints (for example, the power delivered, maximum temperature, etc.). To evaluate a set of parameters, a battery is charged and discharged for as many cycles as possible before it dies. As most batteries cycle for hundreds (or sometimes thousands) of cycles before being obsolete, this results in months of testing. Iteratively sweeping through all possible sets of parameters could take forever. Also, this process incurs the cost of electricity.

## 1.2 Objective of the project

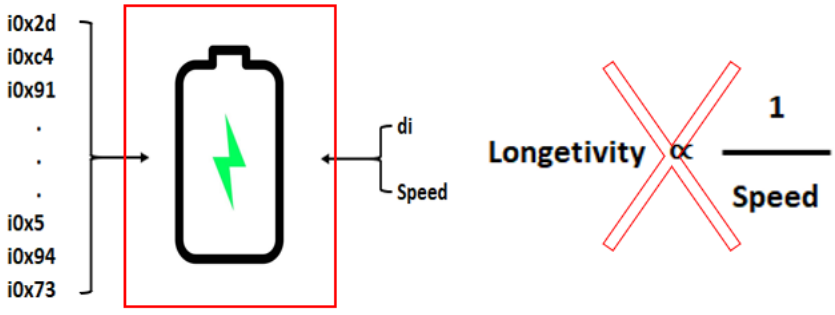
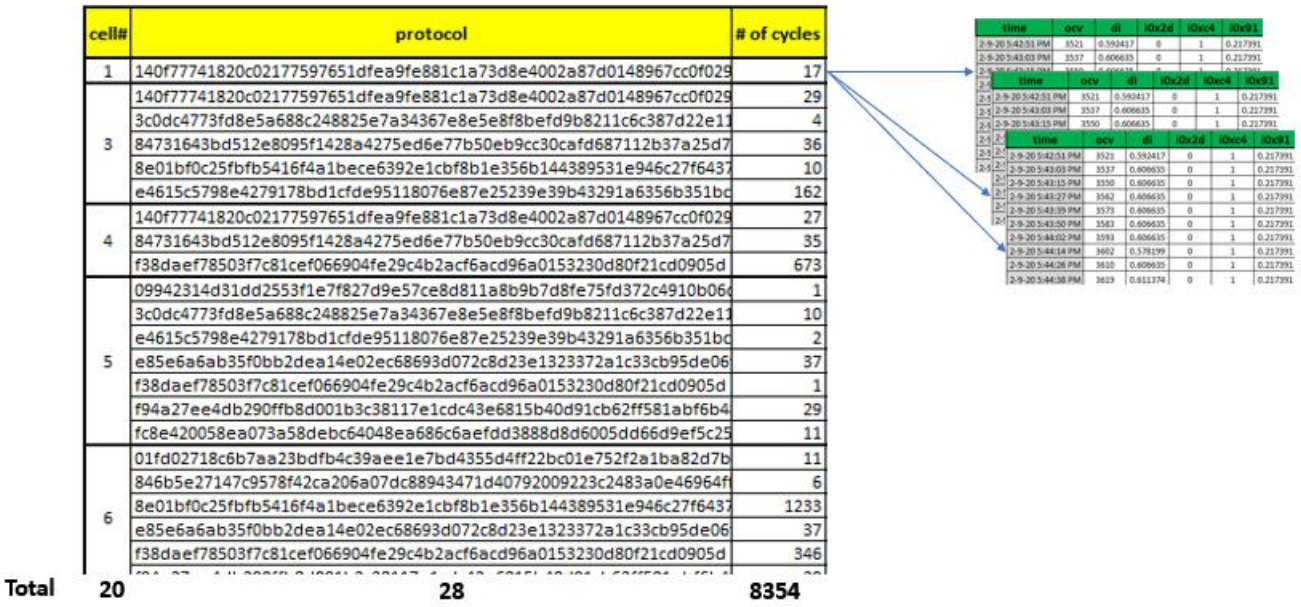



*Figure 1 Input and output of the charging process*

The objective of this project is to leverage the GBatteries' dataset to develop a model of a charging process where inputs are charging parameters and outputs are degradation indicator and speed. The model that explains the functional relationship of degradation indicator and charging speed with charging parameters would provide the most value. An explainable model can be used to form an optimization problem to find the best parameters.

# 2 Dataset



| cell# | protocol | # of cycles |
|---|---|---|
| 1 | 140f77741820c02177597651dfea9fe881c1a73d8e4002a87d0148967cc0f029 | 17 |
| 3 | 140f77741820c02177597651dfea9fe881c1a73d8e4002a87d0148967cc0f029 | 29 |
| | 3c0dc4773fd8e5a688c248825e7a34367e8e5e8f8befd9b8211c6c387d22e11 | 4 |
| | 84731643bd512e8095f1428a4275ed6e77b50eb9cc30cafd687112b37a25d7 | 36 |
| | 8e01bf0c25fbfb5416f4a1bece6392e1cbf8b1e356b144389531e946c27f6437 | 10 |
| | e4615c5798e4279178bd1cfde95118076e87e25239e39b43291a6356b351bc | 162 |
| 4 | 140f77741820c02177597651dfea9fe881c1a73d8e4002a87d0148967cc0f029 | 27 |
| | 84731643bd512e8095f1428a4275ed6e77b50eb9cc30cafd687112b37a25d7 | 35 |
| | f38daef78503f7c81cef066904fe29c4b2acf6acd96a0153230d80f21cd0905d | 673 |
| 5 | 09942314d31dd2553f1e7f827d9e57ce8d811a8b9b7d8fe75fd372c4910b06 | 1 |
| | 3c0dc4773fd8e5a688c248825e7a34367e8e5e8f8befd9b8211c6c387d22e11 | 10 |
| | e4615c5798e4279178bd1cfde95118076e87e25239e39b43291a6356b351bc | 2 |
| | e85e6a6ab35f0bb2dea14e02ec68693d072c8d23e1323372a1c33cb95de06 | 37 |
| | f38daef78503f7c81cef066904fe29c4b2acf6acd96a0153230d80f21cd0905d | 1 |
| | f94a27ee4db290ffb8d001b3c38117e1cdc43e6815b40d91cb62ff581abf6b4 | 29 |
| | fc8e420058ea073a58debc64048ea686c6aefdd3888d8d6005dd66d9ef5c25 | 11 |
| 6 | 01fd02718c6b7aa23bdfb4c39aee1e7bd4355d4ff22bc01e752f2a1ba82d7b | 11 |
| | 846b5e27147c9578f42ca206a07dc88943471d40792009223c2483a0e46964f | 6 |
| | 8e01bf0c25fbfb5416f4a1bece6392e1cbf8b1e356b144389531e946c27f6437 | 1233 |
| | e85e6a6ab35f0bb2dea14e02ec68693d072c8d23e1323372a1c33cb95de06 | 37 |
| | f38daef78503f7c81cef066904fe29c4b2acf6acd96a0153230d80f21cd0905d | 346 |
| Total | 20 | 28 8354 |

*Figure 2 Structure of the data*

Figure 2 summarizes the structure of data. 20 different cells of the same type are used. Each cell is tested for as minimum as 1 or as maximum as 7 different protocols. In total, 28 different protocols are tested. For each cell-protocol combination, data of multiple cycles are recorded. Each cycle has timestamped sensor readings for Open Circuit Voltage (OCV), Degradation indicator (di), and 36 charging parameters (i0x**##**), recorded in a separate CSV file. Figure 3 below shows how data for a single cycle look like.

| time | OCV | di | i0x2d | i0xc4 | | i0x94 | i0x73 |
|---|---|---|---|---|---|---|---|
| 1573352193 | 3531 | 0.56872 | 0 | 1 | | -0.051 | 0.051 |
| 1573352205 | 3538 | 0.57346 | 0 | 1 | ... | -0.042 | 0.053 |
| 1573352217 | 3543 | 0.58293 | 0 | 1 | | -0.032 | 0.052 |

Parameter Labels

- **time:** Epoch UTC **timestamp** in milliseconds
- **OCV:** The measured **open circuit voltage** of the battery (in millivolts)
- **di:** The measured **degradation indicator**
- **i0x##:** Each of these **36 parameters** is a configurable setting in charging protocol

*Figure 3 Data for a single cycle*

## 2.1 Exploratory Data Analysis (EDA)

### 2.1.1 There are primarily three types of features:

1) **Binary features (16)** `['i0x2d', 'i0x81', 'i0x40', 'i0x65', 'i0x30', 'i0x9f', 'i0x6b', 'i0x9', 'i0x3b', 'i0xc9', 'i0xb2', 'i0x14', 'i0x76', 'i0x29', 'i0x2c', 'i0x78']`
2) **Ternary features (6)** `['i0xc4', 'i0x32', 'i0xbc', 'i0x5a', 'i0xb6', 'i0x5']`
3) **Continuous (13)** `[remaining]`

'i0x8f' remains constant throughout all data, and hence it is removed from the features. All the continuous features have values between -1 and 1 (preprocessed).

### 2.1.2 Notable characteristics of features:

- features i0x30, i0x9f, and i0x2c remains constant for a cell-protocol combination. i0x9f, and i0x2c also remain constant for a protocol.
- features 'i0x2d', 'i0xc4', 'i0x81', 'i0x40', 'i0x32', 'i0xbc', 'i0x6b', 'i0x9', 'i0x8f', 'i0x3b', 'i0xc9', 'i0xb2', 'i0x14', 'i0x76', 'i0x29', 'i0x5', 'i0x30', 'i0x9f', 'i0x2c' remains constant within a cycle. (16 cycle specific, 3 which also remain constant for cell-protocol combination)

For ease of understanding while model development, some of the i0x## features are renamed based on their characteristics as follows:

- last two letters are same as ## in the 'i0x##' in the original feature names.
- 'B_' before the last ## means that the feature is binary.
- 'T_' before the last ## means that the feature is Ternary.
- If a feature is prefixed with 'c_const', it remains constant throughout a single cycle.
- If a feature is prefixed with 'P_const', it remains constant throughout a single cell-protocol combination.
- If a feature starts with 'i0x', it varies throughout the data, and it does not have any of the above characteristics.

### 2.1.3 Correlation between features:

The heat map in figure 4 shows the correlation between features. Some features are highly correlated. They are treated appropriately during the feature engineering and selection, as explained in the following sections.

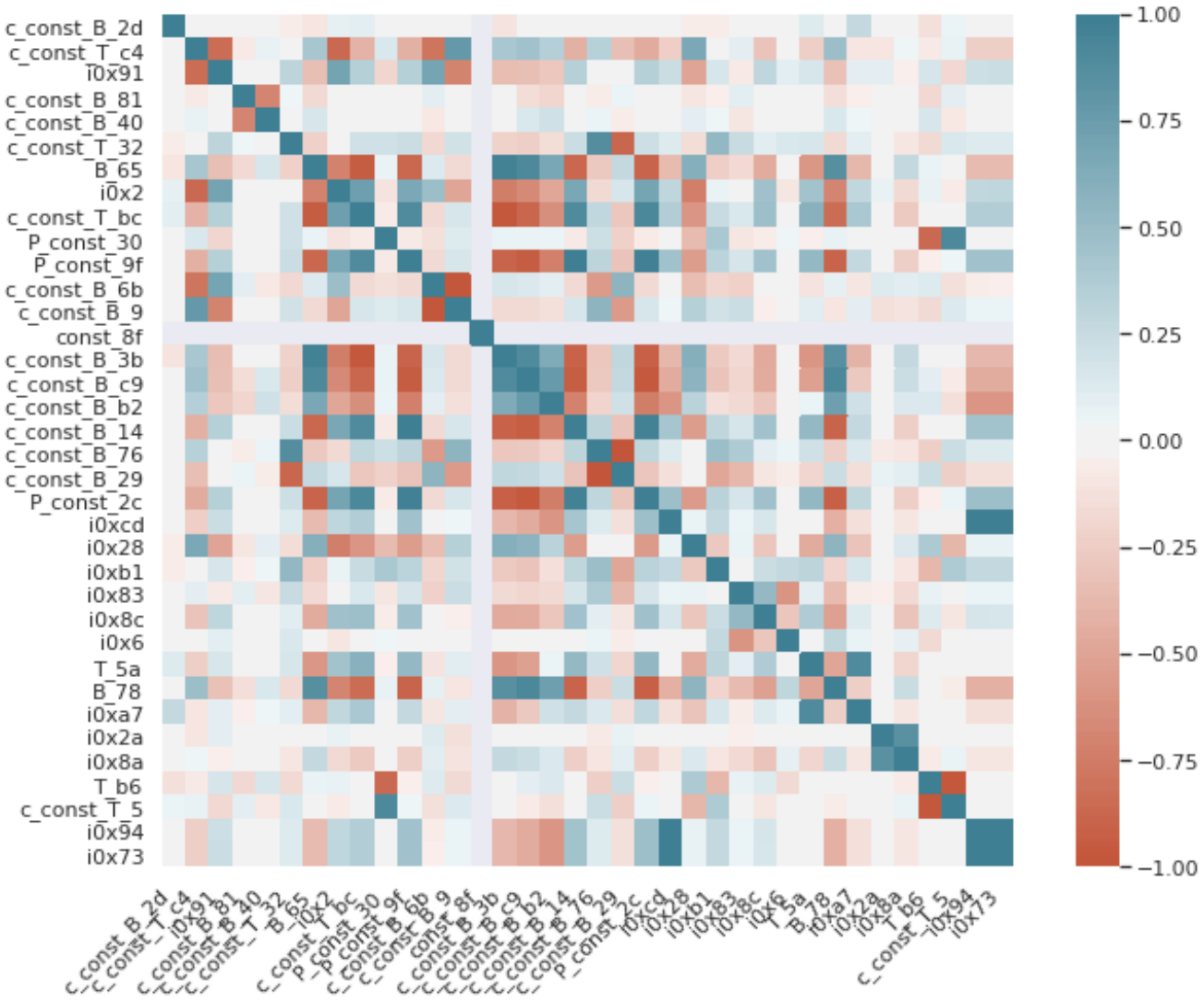



*Figure 4 Heatmap of correlation between features*

# 3 Cleaning & Imputation

Mainly, there are two types of anomalies in the data: missing observations and corrupt observations.

## 3.1 Missing observations

Observations are missing due to three different reasons:

1) Reference Performance Test (RPT): After every 50 cycles during the process, a few slow cycles are performed that allow to measure the usable capacity, internal impedance, and some other metrics to understand the health of a battery. The cycles for RPTs are not included in the dataset. Hence, it appears as missing cycles in the data. These cycles are missing structurally, and they are followed by cycles with a spike in di values. Figure 5 illustrates this fact.

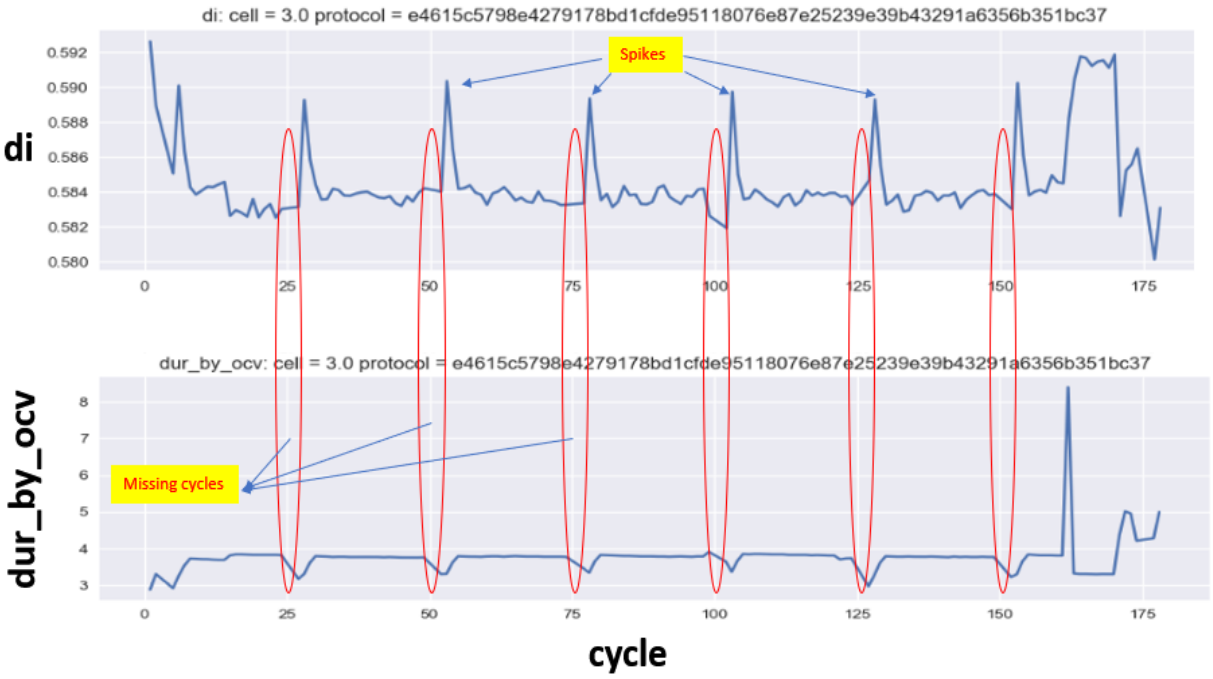



*Figure 5 Missing cycles due to Reference Performance Test*

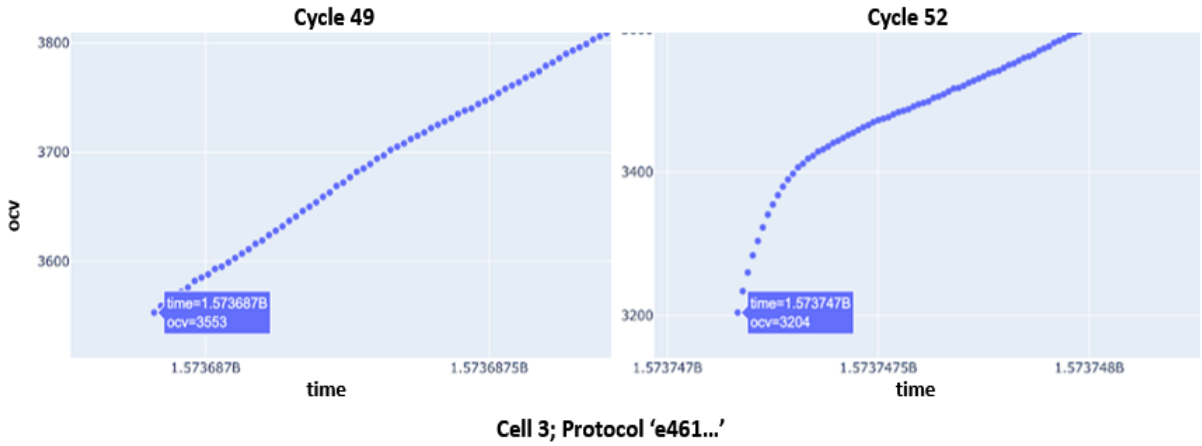



*Figure 6 OCV curve on cycle 49 (before RPTs) and cycle 52 (after RPTs)*

Compared to a regular cycle, the battery is left at a lower voltage at the end of the RPT cycles, as shown in figure 6. When a battery is charged aggressively starting at a lower voltage, it is slightly more damaging. Hence, the spike in the di is observed.
The values of a last regular cycle are forward filled to remove these intermediate missing cycles.

2) Due to the nature of experiments: 4-6 initial cycles are missing for most of the cell-protocol combinations. As these cycles are missing at the beginning, no imputation is done.
3) Corrupt files: some of the cycles are randomly missing in the data due to corrupt CSV files. The values of the last regular cycle are forward filled.

## 3.2 Corrupt observations

These are mainly due to two reasons:

1) Charge and Discharge cycle are mixed: For some cycles, a CSV file contained data for the discharge cycle in it along with the charge cycle. The data of discharge cycles are removed.
2) Instruments kept recording while a cell was not charging: In some cases, the cell kept plugged in even after it is fully charged. Only the additional observations are cleaned for such instances.
For cycle 29 of cell 4, protocol: '140f7…'; all cycles of cell 9, protocol: '0ee15…'; and cell 9, protocol: '6ab1b…', cells did not charge correctly, leading to corrupt observations. The data for these 66 cycles are discarded altogether.

## 3.3 Resampling

For most cycles, observations are taken at regular intervals, making each of those individual cycles a time-series. For some cycles, observations are not taken at a regular interval; and hence, each of those is irregular time-series.

It is tempting to stack all the cycles for a given cell-protocol combination to obtain a big series because cycles themselves follow a temporal pattern. However, there are a few problems with this approach:

1) The time interval between observations varies across cycles and cells, and hence, the resulting time series will not be regular.
2) Each cycle is performed for a slightly different range of OCV. Usually, as the cell gets older, its capacity to retain charge decreases resulting in less charging time. This makes the resulting series difficult to characterize.
3) Even if we model it as a stacked series of individual cycles, we will end with 49 such series, one corresponding to each cell-protocol combination. It is difficult to have a single model that accounts for the effect of exogenous input parameters and generalizes to all the data in such a case.

The first two problems are avoided by downsampling the data to the cycle level. It should be noted that any kind of resampling adds noise to the data. The observations for each cycle are aggregated into one observation per cycle. 19 input features remain constant within each cycle, and hence, their constant values for a given cycle are taken without adding any noise. For the rest of the input features and di, average values are taken while aggregating. To account for the effect of OCV on di (especially for the cycle following the RPT), minimum,

maximum, and range of OCV (3 OCV related features) are included in the data. For a measure of speed, Unit charge time (dur_by_ocv) is calculated by taking ratio of the charge duration and OCV range. The dur_by_ocv can be interpreted as an average time required in seconds for a millivolt of charging. The cycle number is added as a feature to capture the temporal effect between subsequent cycles.

The third problem mentioned above is overcome using dummy features, as explained in the next section, and they are added to the aggregated data.

This downsampled data is referred to as aggregated data or cyc_agg_DF (cycle aggregated data) throughout the report. The models will predict avg di (a measure of degradation) and dur_by_ocv (a measure of speed) for each cycle using this data.

# 4 Feature Engineering and Selection

## 4.1 Fixed Effects

Even after aggregation, we end up with 49 series, one corresponding to each cell-protocol combination. Simply pooling the data and training the model will result in biased estimates of coefficients for regressors (charging parameters). The observations are taken on different cells; each cell may have slightly different physical properties due to manufacturing tolerances. These and other features that are not charging parameters and remain constant for a given cell can lead to omitted variable bias. This is very usual in econometrics and referred to as fixed effects [1]. If the fixed effects due to heterogeneity between different cells are not captured, the errors will not be random. The charging parameters or the output variables will have a correlation with errors, which leads to bias in the estimate of the coefficients.

To avoid this bias, the One Hot encoded dummy variables are created, one for each cell. For example, the dummy variable for cell 1 is 1 when observations are taken on that cell and 0 elsewhere. These features are added to the aggregated data.

## 4.2 Nonlinear combinations and selection

Linear models are easy to interpret, fast to train, and transparent in predictions. However, they do not account for the non-linearity between features and predicted variables. To overcome this, nonlinear features are created using the following transformations:

**`1/, exp, log, abs, sqrt, ^2, ^3, 1+, 1-, sin, cos, exp-, 2^`**

Furthermore, these and original features are combined (addition, subtraction, multiplication) and again transformed to create additional features. Then, model-based multivariate feature selection is performed on the final feature set.

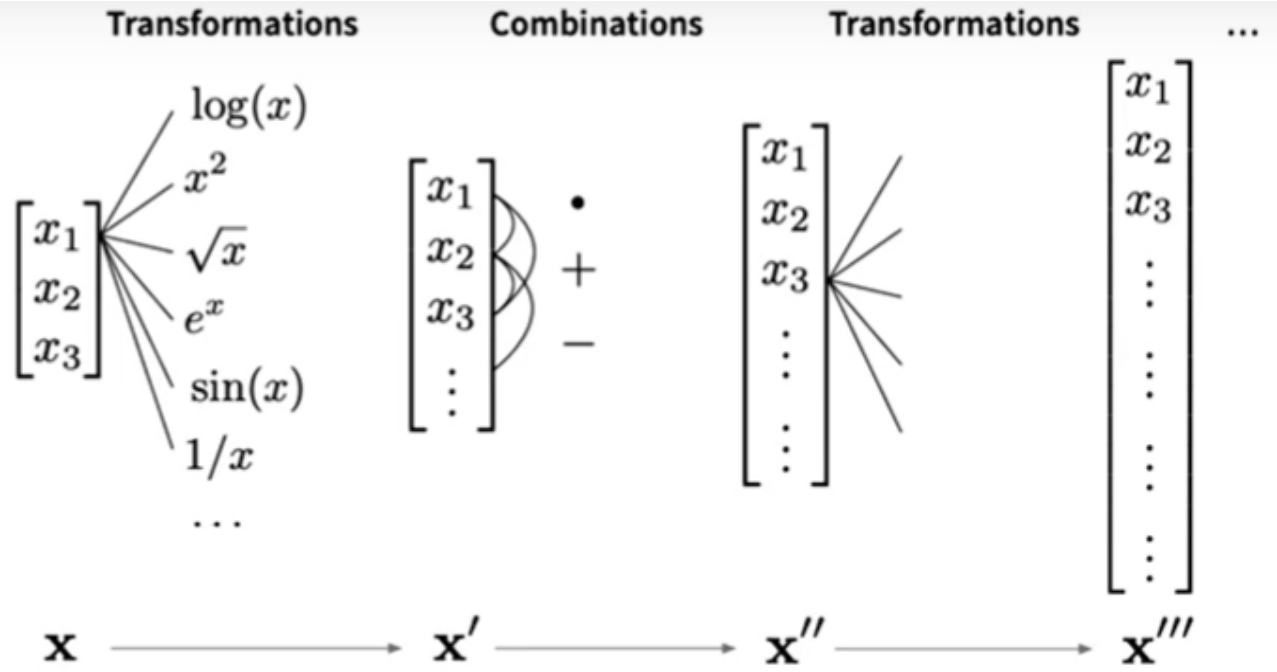



*Figure 7 feature generation and transformation stages (courtesy: Franziska Horn)*

Features from the new set are added to the original feature set iteratively based on their correlation with the residual of target variables. On each iteration, a feature with the highest correlation with residual is added, and the model is retrained with an updated feature set to find the new residual. This process is repeated until convergence. To avoid overfitting, a different subset of data is used for each iteration. The features with very low coefficients are discarded during the process. After the convergence, noise filtering is performed. For that, features with random values are created. A Lasso model is trained on selected and random features. The features which are less significant than random features are discarded.

To scale and automate this task, a python library Autofeat [2] is used.

After carrying out this process, we obtained 60 features for di and 70 features for dur_by_ocv (including dummies for fixed effects).

# 5 Train-Test split

To avoid the leakage of information from data of future cycles, all the data is split in a deterministic manner. For all cell-protocol combinations, the first 80% of the cycles are used for the train-validation set. The remaining last 20% cycles are used for the test set.

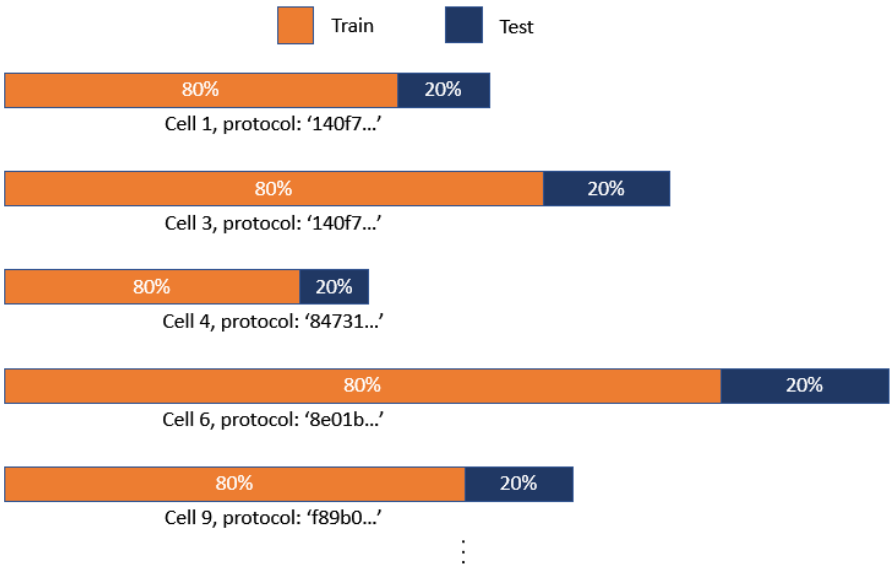



*Figure 8 Train-Test Split*

It should be noted that the first 3 models explained below do not use any lag features, and hence, a random split can also be used in that case without additional care.

# 6 Models

## 6.1 Linear Regression (L2, Fixed effects)

Linear models with L2 regularization are developed using aggregated data (cyc_agg_DF) and feature engineered data (FE_cyc_agg_DF) for both di and dur_by_ocv. A regularization parameter is tuned using K-fold cross-validation. For spark ML, its value is 0.00005. This may vary for different libraries depending on the loss function definition.

### 6.1.1 Fixed Effects

As explained in feature engineering, dummy variables are included in the feature sets. During the training, the first dummy variable is dropped to avoid multicollinearity, and the model is fitted with intercept. In general, this approach could be interpreted as fitting the different intercept for each cell. Coefficients of dummy variables capture how the corresponding cell intercept is different from the first cell. They are the measure of heterogeneity among the cells.

$$y_{it} = \alpha_i^* + \beta x_{it} + u_{it}, (u_{it}|\boldsymbol{X}) \sim \text{IID}(0, \sigma^2)$$

Individual-specific intercepts, common coefficients

Intercept

$$\alpha_i = \alpha_1 + \alpha_2 \text{D2} + \alpha_3 \text{D3} + \ \ldots \ + \alpha_n \text{Dn}$$

1 if cell# = 2 , else 0
(One Hot Encoding)

### 6.1.2 Performance

Tables in Figure 9 below summaries the performance of all four models: 2 for di and 2 for dur_by_ocv. (first using aggregated data and second using feature engineered data for both di and dur_by_ocv). Coefficient of determination ($r^2$), Mean Squared Error, and symmetric Mean Absolute Percentage Errors are standard matrices used to measure the performance of the regression models. In addition, predicted $r^2$ is also used.

**Predicted $r^2$ (PRESS R-squared)**

It is a more statistically efficient estimate for generalization error or goodness-of-fit. It adjusts the estimate of how much variation the model explains by using n-fold cross-validation rather than adjusting for the model's degrees of freedom (as the more standard adjusted R-square does); it tries to estimate how the model will perform on hold-out or new data, using only in-sample data. One observation is hold out, and the model is trained on the rest of the training data. Then the error is measured on the hold-out observation. This process is repeated for all the observations (and hence n-fold cv). Finally, these errors are used to calculate $r^2$.

**a) Original features + Dummies**

| Metrics | di | dur_by_ocv |
|---|---|---|
| $R^2$ on test set | 0.86 | 0.84 |
| Predicted $R^2$ | 0.90 | 0.88 |
| RMSE | 0.011 | 0.494 |
| sMAPE | 1.29% | 6.62% |

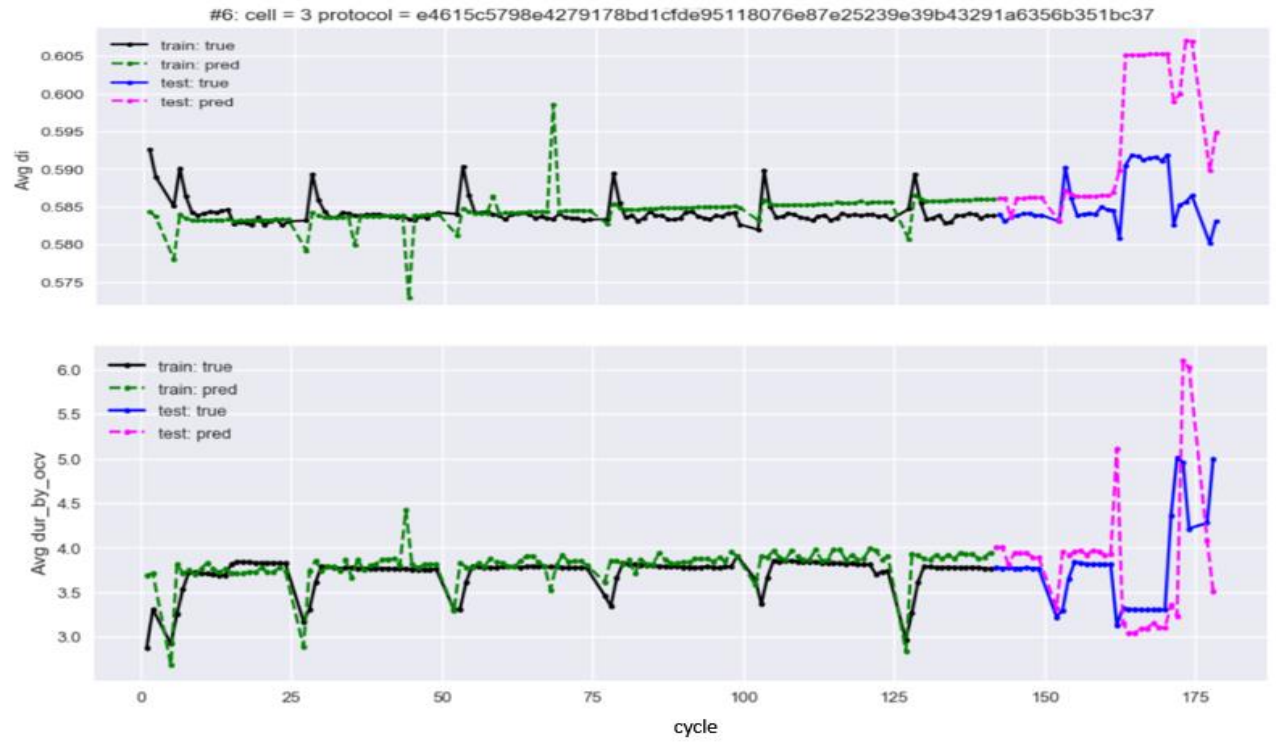
**b) Engineered Features + Dummies**

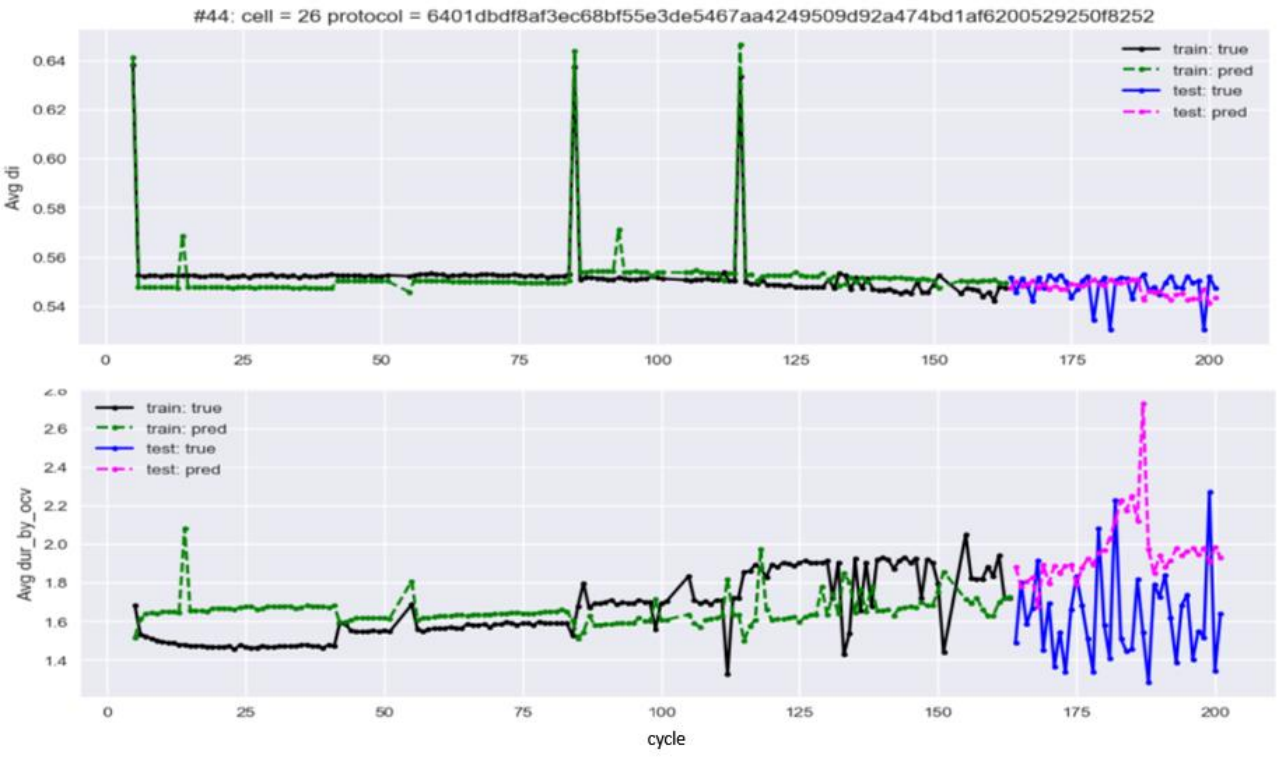
| Metrics | di | dur_by_ocv |
|---|---|---|
| $R^2$ on test set | 0.898 | 0.92 |
| Predicted $R^2$ | 0.92 | 0.95 |
| RMSE | 0.010 | 0.341 |
| sMAPE | 0.74% | 5.18% |

*Figure 9 Performance of linear regression models*

The engineered features improved the generalization of the model. R2 increase from 0.84 to 0.92 for dur_by_ocv and 0.86 to 0.898 for di. Overall, models perform well for aggregated data.

The graphs below plot the predicted and true values for 3 of the 49 different cell-protocol combinations for di and dur_by_ocv. The models captured the overall trend of the data well. Also, the models are more accurate for the cell-protocol combinations with more than 100 cycles of data. The performance is relatively poor for cell-protocol combinations with fewer data.



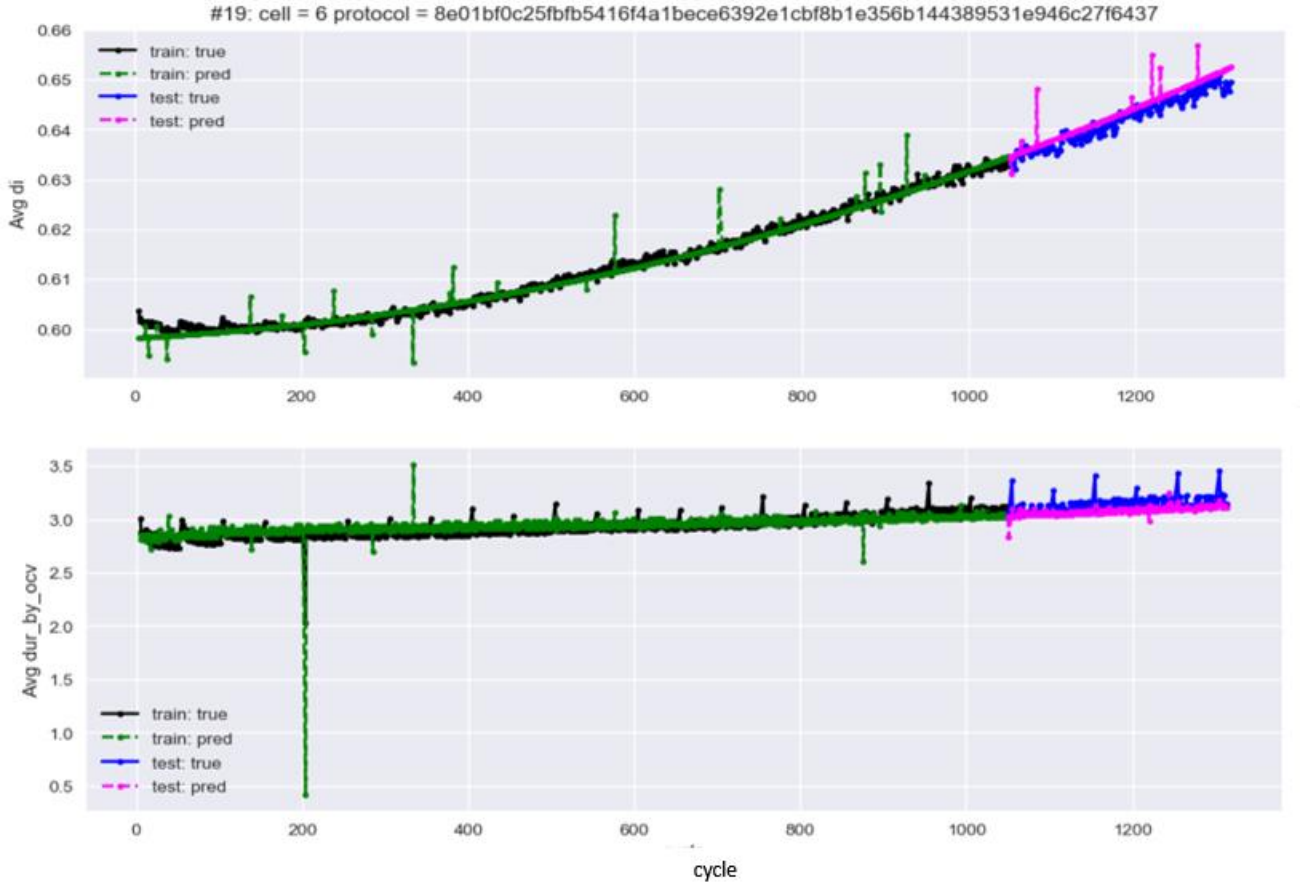



*Figure 10 fit of linear regression models for di and OCV for three typical cell-protocol combinations*

## 6.2 Random Forest (RF)

Random forest is an ensemble of multiple regression trees. It outputs the average of predictions of individual trees. Bagging in random forest reduces the variance (overfit) of the model. It does not require a lot of pre-processing for features. The RF has two hyperparameters, and it is significantly less sensitive to them.

### 6.2.1 Optimal models

The hyperparameters are tuned using the parameter grid and cross-validation. The best model for di has the maximum number of trees 50 and maximum depth 25 after tuning. Similarly, for dur_by_ocv, these values are 10 and 5, respectively.

### 6.2.2 Performance

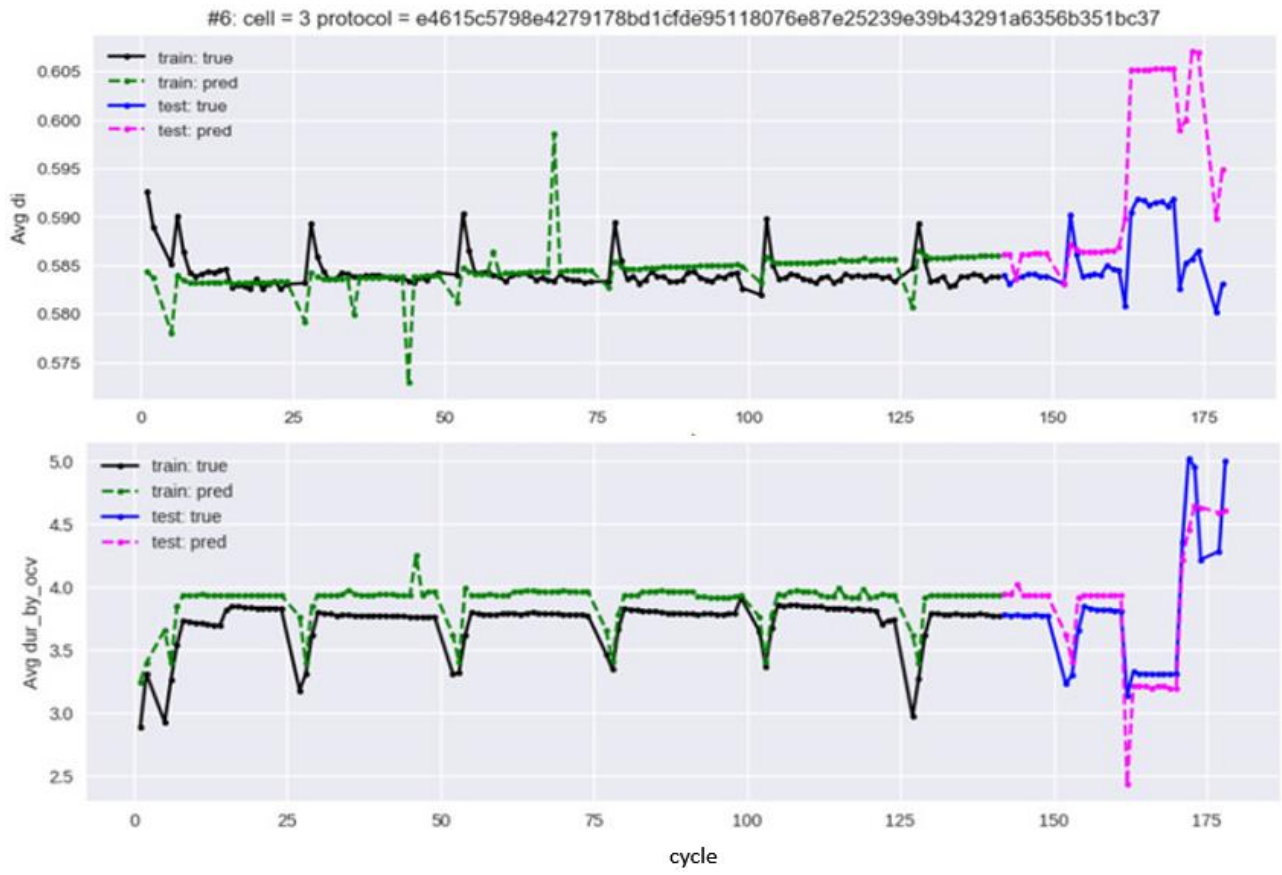
The table in figure 11 summarizes the model performance.

**a) Original features + Dummies**

| Metrics | di | dur_by_ocv |
|---|---|---|
| $R^2$ on test set | 0.74 | 0.91 |
| RMSE | 0.015 | 0.36 |
| sMAPE | 2.24% | 6.81% |

*Figure 11 Performance of Random Forest models*

The graphs below show the predicted and true values for three of the 49 different cell-protocol combinations.



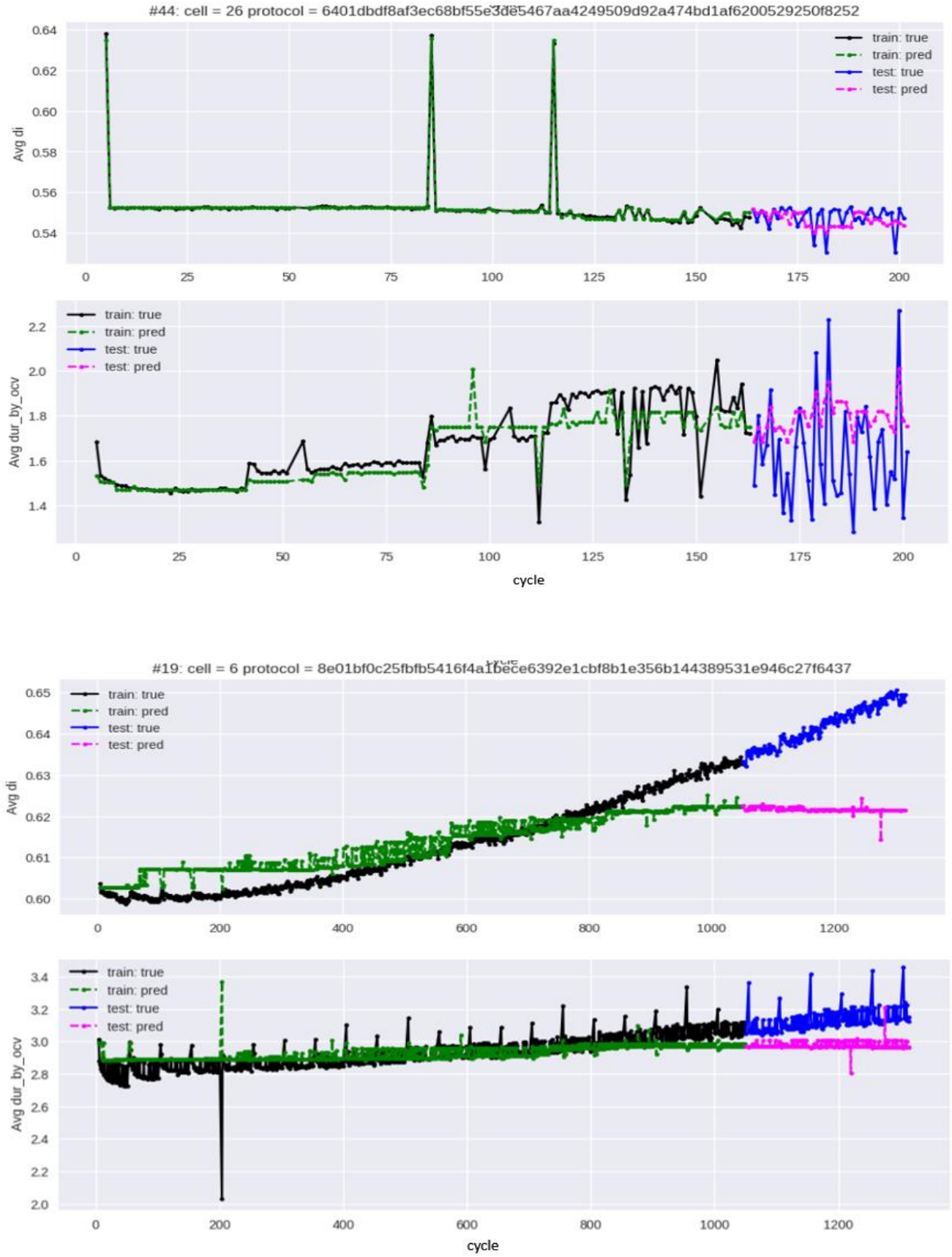



*Figure 12 fit of Random Forest models for di and OCV for three typical cell-protocol combinations*

### 6.2.3 Feature Importance

The pie charts below show the feature importance for di and dur_by_ocv.

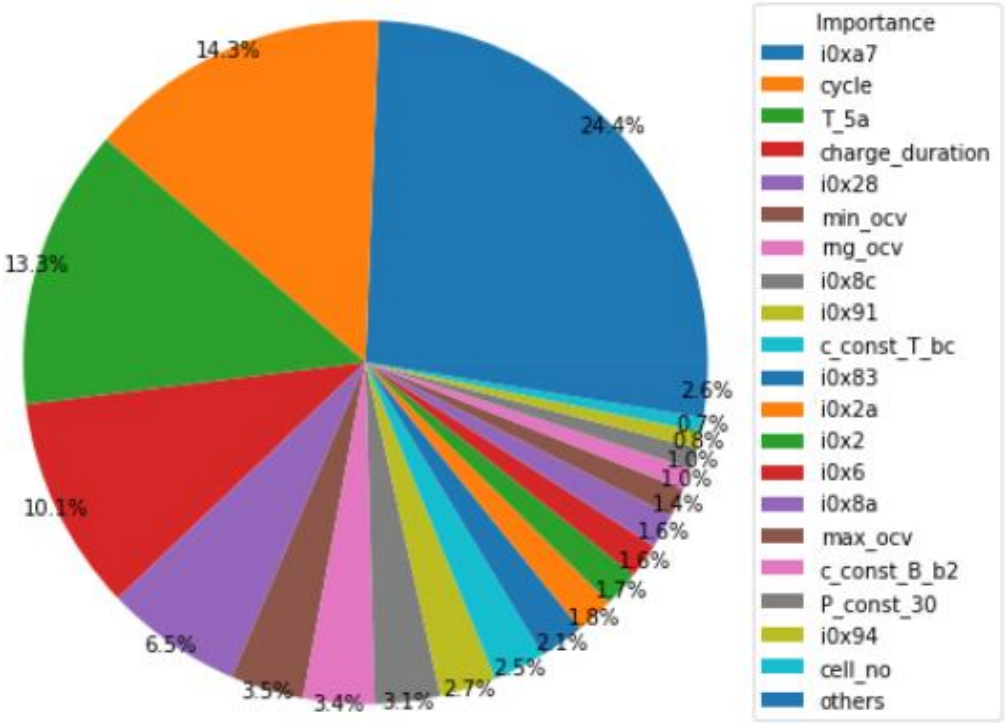



*Figure 13 Feature Importance for di (Random Forest)*

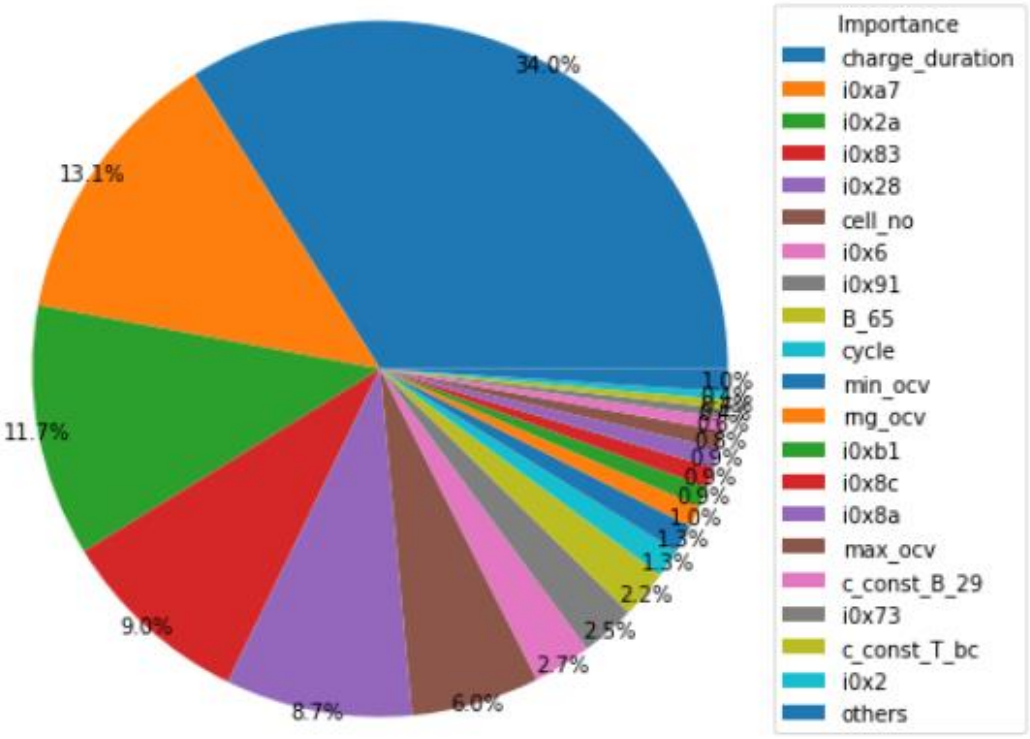



*Figure 14 Feature Importance for dur_by_ocv (Random Forest)*

## 6.3 Gradient Boosted Trees (GBT)

GBT is one most effective machine learning algorithms. It minimizes both bias and variance of the model. GBT train one tree at a time, where each new tree helps to correct errors made by previously trained trees. With each tree added, the model becomes even more expressive.

GBT can be viewed as iterative functional gradient descent algorithms. That is, it optimizes a cost function over a function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

### 6.3.1 Optimal models

The best model has the maximum number of trees 30 and the maximum depth 10 after tuning.

### 6.3.2 Performance

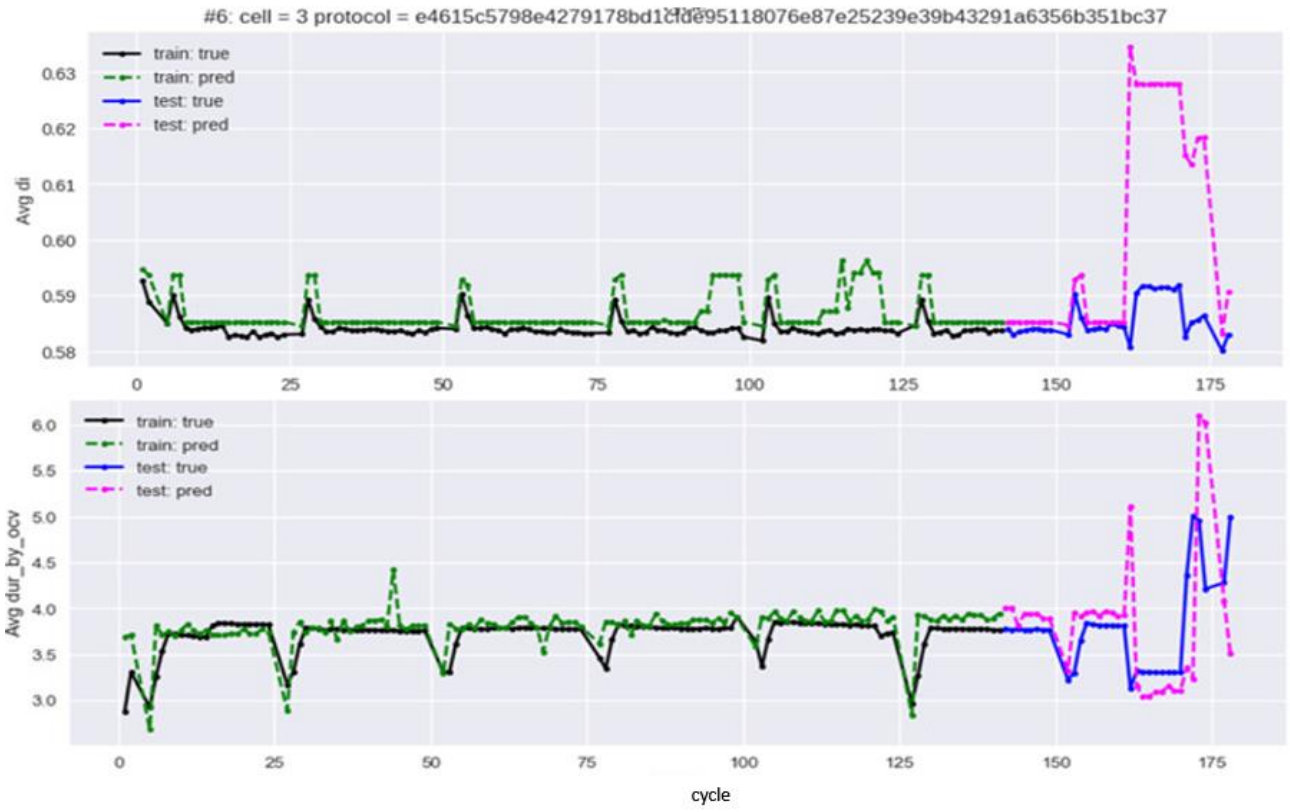
The table in figure 15 summarizes the model performance.

**a) Original features + Dummies**

| Metrics | di | dur_by_ocv |
|---|---|---|
| $R^2$ on test set | 0.93 | 0.92 |
| RMSE | 0.008 | 0.35 |
| sMAPE | 0.87% | 4.21% |

*Figure 15 Performance of Gradient Boosted Trees models*

The graphs below plot the predicted and true values for three of the 49 different cell-protocol combinations.



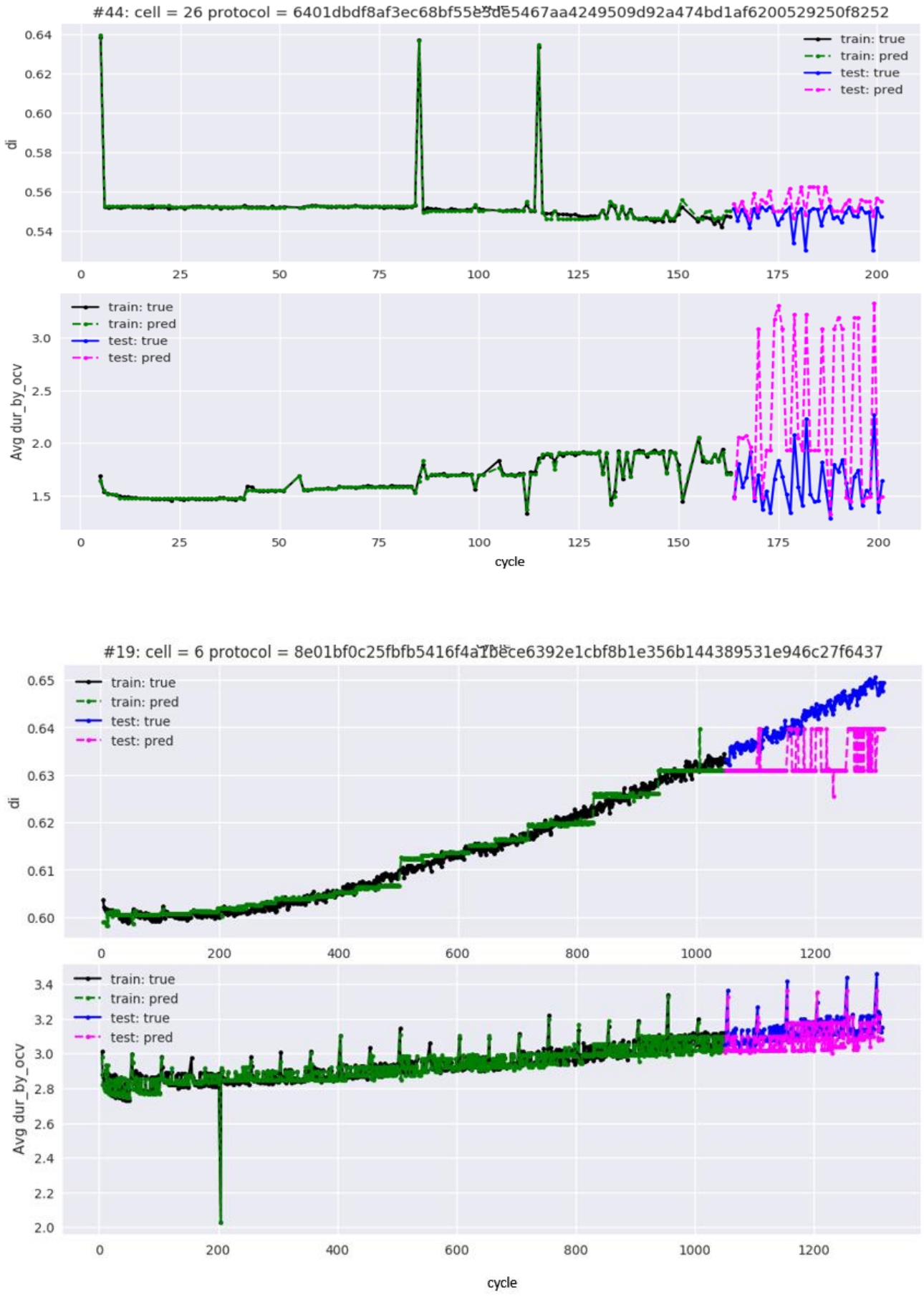



*Figure 16 fit of GBT models for di and OCV for three typical cell-protocol combinations*

### 6.3.3 Feature Importance

The pie chart below shows the feature importance for di and dur_by_ocv.

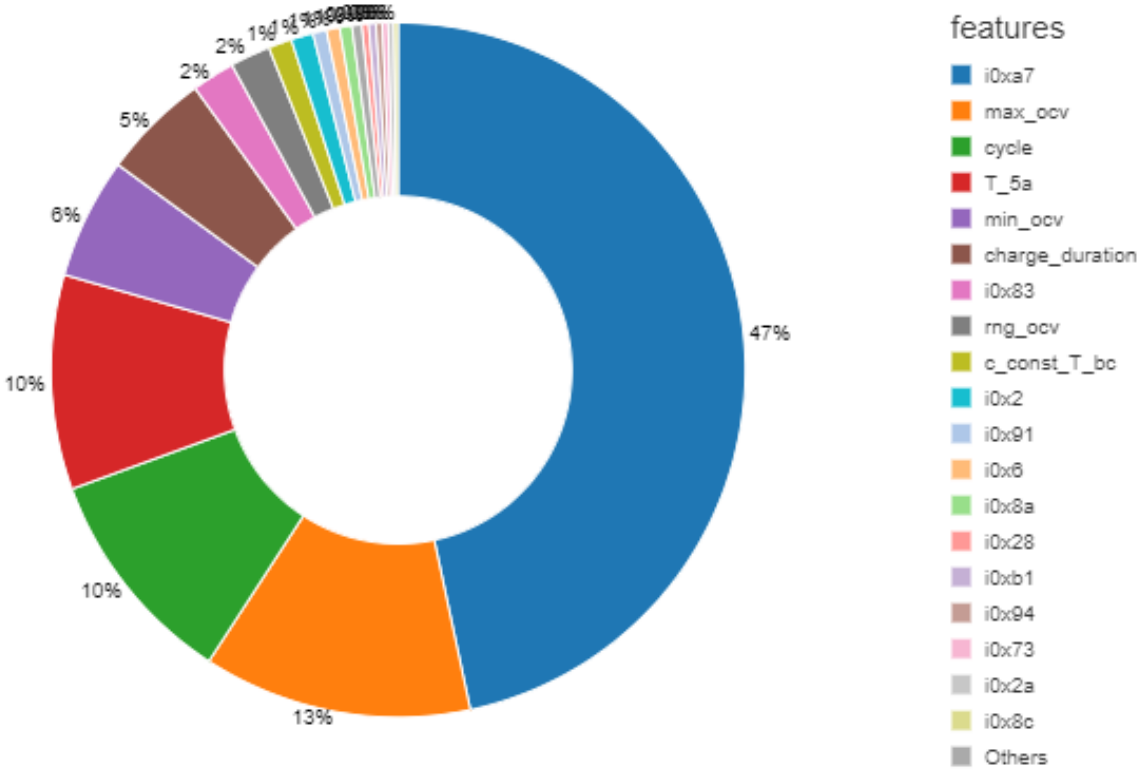



*Figure 17 Feature Importance for di (Gradient Boosted Trees)*

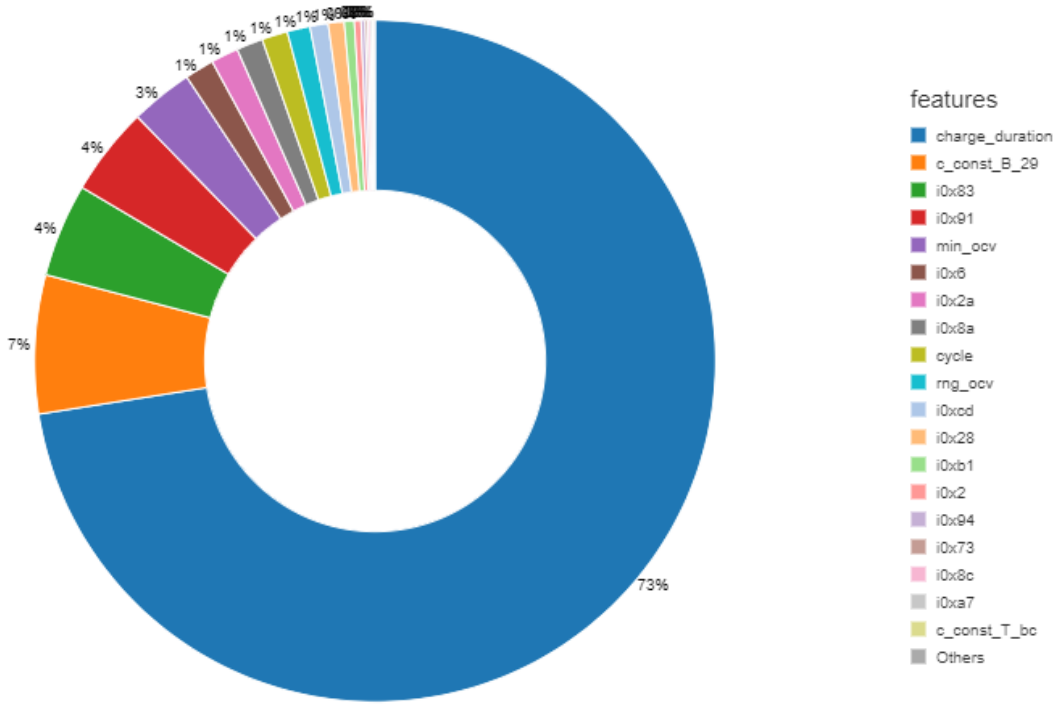



*Figure 18 Feature Importance for dur_by_ocv (Gradient Boosted Trees)*

# 7 Comparison of models

The table below summarizes the performance of the best models for each algorithm.

| ALGORITHMS | METRICS | | | | | |
|---|---|---|---|---|---|---|
| | RMSE | | R2 | | SMAPE | |
| | di | dur_by_ocv | di | dur_by_ocv | di | dur_by_ocv |
| **1.Linear Regression** | **0.011** | **0.494** | **0.86** | **0.84** | **1.29** | **6.27** |
| **2.Random Forest Trees** | **0.015** | **0.36** | **0.74** | **0.91** | **2.24** | **6.81** |
| **3.Gradient Boosting Trees** | **0.008** | **0.35** | **0.93** | **0.92** | **0.87** | **4.21** |
| **4.Linear Regression (with engineered features)** | **0.010** | **0.341** | **0.898** | **0.924** | **0.74** | **5.18** |

Major points:

1) GBT is more effective in terms of learning from features. It did not require any feature engineering to improve performance. In fact, the model trained with engineered features gave the same results as the model with only aggregated original data. However, it took approximately 2 hours to tune hyperparameters for GBT.
2) Linear regression with engineered features matched the performance of GBT. It took almost the same time for automated feature engineering as it took to tune the GBT.
3) The time required for hyperparameter tuning and/or feature engineering is not a problem because we only need to do them once.
4) However, when we want to repeatedly train the model in production with new data, the linear regression with engineered feature would be a better option. It is fast compared to GBT during training.
5) Also, linear regression is fast in terms of hyperparameter tuning—only one parameter to tune when using regularization or none in case of simple linear regression.
6) GBT can be used to benchmark the success of feature engineering. Once a good set of engineered features is obtained that give similar performance to GBT with a linear model, the linear model with engineered features can be deployed in production.
7) The linear regression model is easy to interpret. It can be used to form an optimization problem for input features.
8) The results of GBT and Linear regression can be combined to understand the importance of a feature and its functional relationship with output variables.

The pie charts below show the importance of features for GBT models trained using engineered features. For the top features, their functional relationships with output variables are obtained from linear regression models trained on the same data. This information allows us to selectively minimize or maximize features' values based on their effect on output. The amount of change in the output can also be quantified from linear regression models.

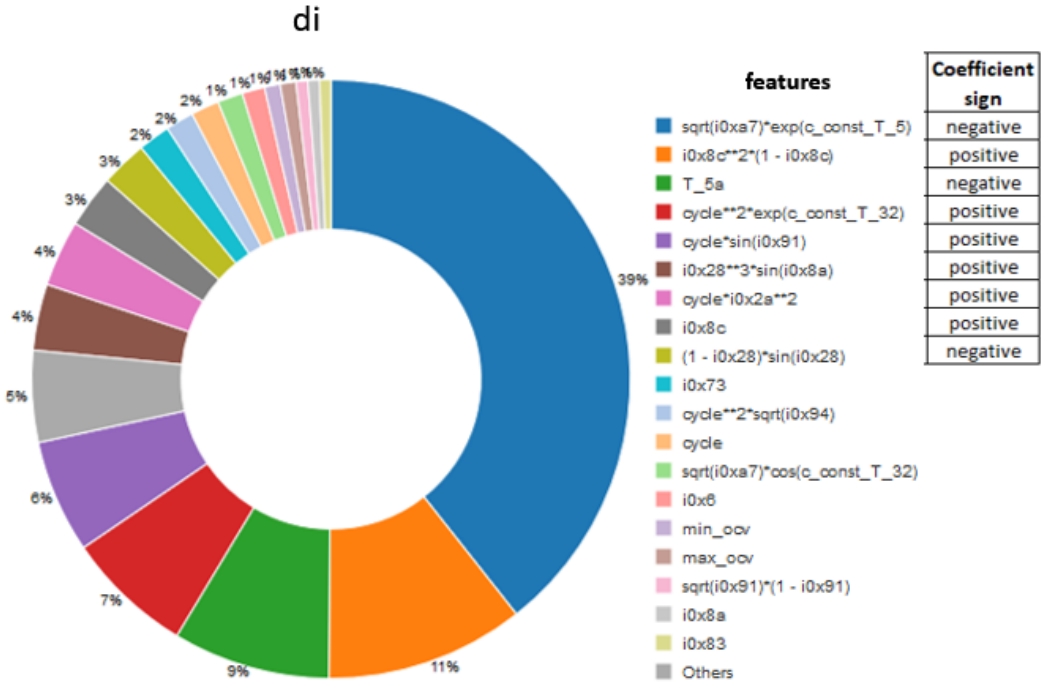



*Figure 19 Feature importance for GBT and their effect on output from Linear Regression for di (Engineered Features)*

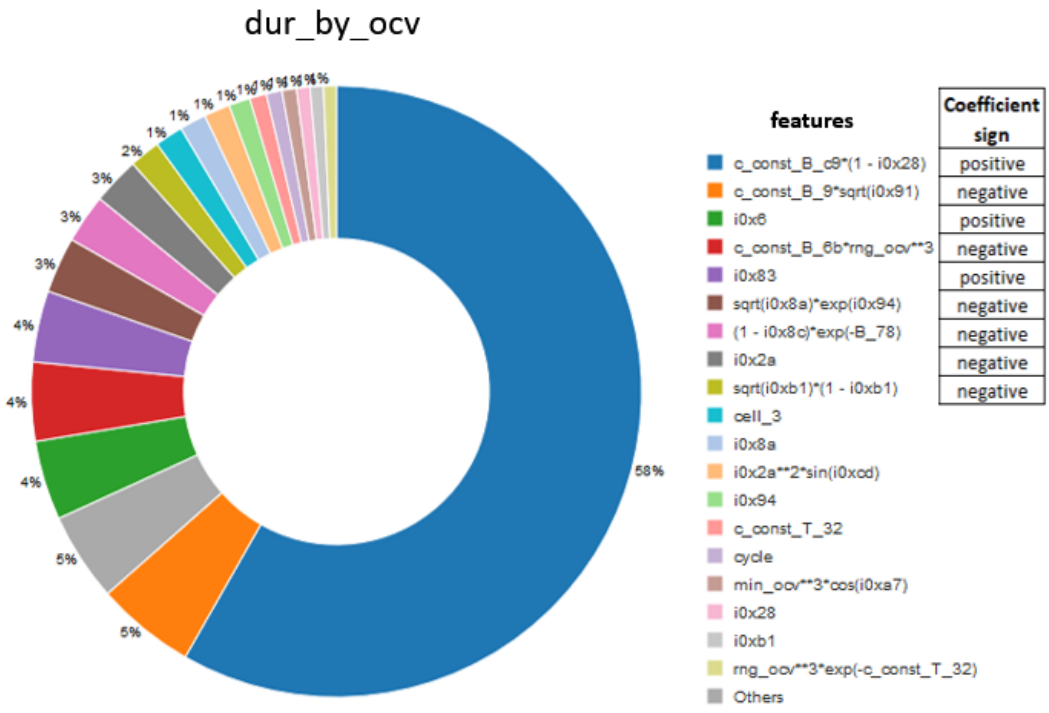



*Figure 20 Feature importance for GBT and their effect on output from Linear Regression for dur_by_ocv (Engineered Features)*

# 8 Scope of Future Work

At the beginning of the project, we tried the statistical models such as ARIMA (Auto-Regressive Integrated Moving Average) to capture the effect of previous values (lags) of di and dur_by_ocv on their future values. We eventually gave up on those models, mainly due to panel behaviour and irregularity in the data:

1) Irregularity: All the widely use stochastic models such as ARIMAX, GARCH, VAR requires data to be regularly observed. As GBatteries actively collects new data, one of the following could be done:
    a. Sensor reading can be observed at a regular predefined interval (for example, every 60 seconds). The interval can be chosen based on the granularity desired.
    b. Observations can be taken at predefined OCV values. For example, after every 50 millivolts of charging, a new observation can be recorded, starting from a predefined OCV value. Not only did this remove the irregularity within a cycle, but it will also remove irregularity between cycles due to the change in charge capacity of a battery. The current charging time can be treated as just another parameter, as every observation will be indexed by OCV values.

    As the underlying charging process is continuous, regularly taken observations solves the issue with irregularity.

2) Panel behaviour: As the observations are taken on 20 different cells, resulting in 49 different series, one for each cell-protocol combination, the data can not be pulled and fed to the existing implementation of statistical models. First, Spark does not have a standard library for time series. Second, even though python has very well written libraries for time series analysis, those are not implemented for panel data. Using the pooled data and creating lag features would result in incorrect models. The following could be done:
    a. Manually create lag features for each individual series.
    b. Manually implement statistical models to use data with manually created lag features.

In all models mentioned in this report, we used down-sampled data to get rid of irregularity. Resampling introduces the noise. Techniques such as Lomb-Scargle Periodograms could be used to deal with irregularity.

Also, the models such as XGBoost would be an ideal way to move forward with the same data. Neural Network models such as RNN or LSTM could be implemented on less severely resampled data.

During the feature engineering, the dimensionality aspect is completely overlooked. This is because the information about features is concealed for confidentiality. Consequently, some engineered features may not make sense. However, the Autofeat python library can accept physical units of features and create physically meaningful transformations. Also, it can create dimensionless groups (feature combinations) based on the Buckingham Pi theorem used in the dimensional analysis in many branches of engineering. This could ultimately result in a few more meaningful features.

# 9 Conclusion

The work carried out in this project will guide the experiment design in terms of the collection of new data. In addition, the models developed in the project are all explainable. They provide insights into the features, especially their importance and relationship with output variables. Also, they are good predictors of average di and charging speed for individual cycles. These models can be used as a reference to compare the performance of future models.

# Reference

[1] Biørn, E. (2016). *Econometrics of panel data: Methods and applications*. Oxford University Press.

[2] Horn, F., Pack, R., & Rieger, M. (2019, September). The autofeat Python Library for Automated Feature Engineering and Selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 111-120). Springer, Cham.