Proceedings of the ASME 2021 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2021
Virtual Conference, August 17-20, 2021

DETC2021-69647

USING MACHINE LEARNING FOR THE CLASSIFICATION OF THE REMAINING USEFUL CYCLES IN LITHIUM-ION BATTERIES

Harry Coutts
Department of Engineering
Durham University
Durham, United Kingdom

Qing Wang
Department of Engineering
Durham University
Durham, United Kingdom

ABSTRACT

In order to keep up with the increasing focus on renewable energy, the demand for new battery technology and peripherals has likewise increased greatly. Given the relatively slow rate of change of new battery chemistry and technology, it is the peripherals to the batteries that are often relied upon to provide this necessary increase in performance. The 18650 battery with Lithium-Ion internal chemistry is one of the most widely used batteries and is depended upon in many industries to provide power portability and storage. Using an extensive freely available dataset compromising of the charge cycles of 121 18650 batteries, this paper evaluates multiple algorithms' effectiveness at predicting the remaining useful cycles of a battery from a single discharge curve. Upon evaluation of the algorithms, 'Weighted K Nearest Neighbours' was shown to be the most accurate model and was further improved to ensure that the maximum accuracy was acquired. Finally, a user interface was created to allow for the demonstration of a potential use case for the model. This model and user interface show the potential for easy testing of batteries to determine the number of remaining useful cycles. This makes the possibility of re-purposing or extending the initial purpose of these batteries much greater, which is preferable from both an economic standpoint and an ecological one.

1. INTRODUCTION

As the world becomes more focused on the importance of renewable energy generation and consumption in response to the threat of climate change, the reliance on batteries has also increased. This is predominantly due to the move away from non-renewable sources of energy, such as oil, towards renewable sources like wind or solar. In order to make best use of the efficiency of large scale renewable energy generation, the market increasingly relies on batteries to provide portability. Chief among this adoption of battery technology is the automotive industry, a key driver in the expansion of the battery market.

As the electric vehicle (EV) market grows, so does the need for batteries and new battery technology. Specifically, the EV market predominantly uses Lithium-Ion (LI) battery technology within its vehicles as this chemistry provides the greatest benefits, both in terms of manufacturing scale, general availability, and specific energy density [1]. Wagner [2] predicts that the demand for batteries will increase from 184 GWhr of capacity needed in 2018 to 2623 GWhr in 2030, an increase predominantly caused by a 2191 GWhr increase in demand in the electric mobility sector.

As mentioned previously, the most commonly used technology available today is based on a LI internal chemistry [3] which was initially commercialised in the early 1990s [4]. One of the most commonly used battery types is the '18650', a rechargeable LI-based battery that is used in everything from flashlights to the Tesla model 'S' and 'X' cars, where (in the Model S) 7,104 18650 batteries are used to power the car. In this paper the 18650 battery referred to is the APR18650M1A produced by A123 Systems [5], a commercially available LI-based rechargeable battery.

With this slow progress the industry has turned to other ways to maximise the capacity and usefulness of batteries. One solution is to ensure that the batteries are efficiently recycled to reuse any of the metals contained within. Alternatively, there is a large potential for repurposing batteries that are no longer fit for their current use. Whilst these end of life operations are essential in extending the useful life of a battery, it is also important to effectively utilise the battery throughout its 'first life' or initial intended use. Han et al.'s paper [6] splits the design of the battery into four stages: the material, electrode, cell, and system levels, and explains how the design choices at each stage can affect the lifespan of the battery.

The work within this paper will lie within the system level of the battery, which predominantly refers to the mechanical, electrical, and thermal related issues of extending the life of the battery. A large amount of research is being put towards the

battery management system that surrounds the battery, to produce both diagnostic and prognostic evaluations of battery health. This paper will focus on the prediction of the batteries' remaining useful cycles (RUC), which is defined as the number of cycles left until the battery is no longer fit for its initial purpose. Other work in this area includes the use of an Unscented Kalman filter (UKF) alongside a non-linear time series prediction model by Zheng et al. [7] to predict RUL. In the development of this method Zheng et al. applies it to the RUL as well as the short-term capacity prediction of batteries. Alternatively, support-vector machine methods can be used, as demonstrated by Wang et al. [8]. In this case working temperature and energy efficiency are used as inputs to characterise the training dataset. Finally, Wang et al. [9] have developed a regression vector machine (RVM) and combined it with a battery degradation model. They use RVM to select the significant training vector in order to improve the performance of the prediction.

This paper will partly build upon the work done by Severson et al. [10] who used data driven methods to predict battery cycle life before degradation. Using a large publicly available dataset created by Severson et al. [10], this paper details the work done to create and test a viable tool for predicting the remaining useful cycles of a LI battery. MATLAB was chosen as the preferred language for this work due to its multiple features that enable the easy processing of complex and noisy data as well as the built-in applications it provides for both the creation of classification algorithms and a user interface (UI). By utilizing the MATLAB classification learner [11] it was possible to test multiple classification algorithms to determine the most accurate algorithm for this scenario. This was chosen as an avenue of research due to the lack of comparative work done in the field, and while there was potential for loss in accuracy due to a broader overview, it was felt that the benefits from the comparison would be greater. From here, using the exported algorithm a UI was created that allows for the demonstration of the algorithm in a practical use case.

2. THEORY

2.1 Battery Degradation

In an ideal case, there would be no ageing within the battery and the capacity would remain constant, irrespective of the number of times the battery is cycled. Obviously there is no ideal battery and factors such as loss of anode/cathode active material and loss of the electrolyte can cause a decrease in both the capacity and the power of the battery. This paper uses capacity fade as the metric for determining when the battery is no longer fit for purpose.

Numerous factors affect the rate at which a battery degrades, some of which contrast general public belief about what contributes to the ageing of batteries. Chief among this confusion is the depth of discharge (DoD). Previously, with Nickel-based chemistries batteries had 'memories' and as such needed regular 100% discharges to maintain their capacity. This is untrue with LI batteries, in fact full discharges cause the battery life span to drop significantly[12].

The temperature that the battery is stored at can also have an effect on the battery capacity, especially when stored for some time at an elevated temperature. Higher temperatures can accelerate side reactions within the battery causing layers to form on the electrolyte interface that can affect the efficiency of the battery. Conversely, low temperatures can slow down the transport of lithium ions, and attempts at fast charging at low temperatures can cause crowding of lithium ions, which can potentially lead to an internal short circuit.

The last mechanism that can cause a reduction in the life cycle of a battery is the voltage that it is charged to, roughly speaking a reduction of 0.1V/Cell [12] in charge level can have the effect of doubling the cycle life. By overcharging the battery, irreversible damage can be caused to both the anode and the cathode. Conversely to this, a reduction in charge level will lead to a reduction in the capacity the battery holds, with a 70mV reduction in charge level reducing the capacity by 10%. This leads to the need for a trade-off between the desire to fully charge a battery and the need for longevity in charge cycle. Typically speaking, consumer products will emphasise the need for full capacity at the expense of a diminished cycle life.

2.2 Remaining Useful Cycles Calculation

The remaining useful life (RUL) of a battery is defined as the number of cycles it is able to complete before the capacity degrades to a point whereby it is no longer suitable for its initial task; at this point, as discussed earlier, the battery can be recycled or repurposed. Xing et al. [13] also defines the remaining useful life (RUL) as the remaining time or number of cycles before the battery's state of health (SOH) reaches 0%. Estimating the RUL of a non-linear system such as LI batteries is a complex task, but one that is critical for technology development and efficient management of battery systems. Due to the variable nature of battery operations, factors such as dynamic ageing mechanisms, large device variability, and changeable operating conditions, the process of estimating RUL has remained challenging. Despite the important nature of RUL estimation there has never been a universal best practice for its estimation, and as such, the current methodologies differ significantly. Current methods can be broadly categorised as follows: [14] adaptive filter techniques (AFT), intelligent techniques (IT), stochastic techniques (ST), and miscellaneous techniques (MT). The following sections will go into some detail about the surrounding technologies that have been proposed for RUL calculation to provide an overview of the work that has been done in this field.

1) Adaptive Filter Technique: An adaptive filter is a digital filter where the coefficients change with the aim of making the filter converge to an optimal state. The criterion for optimisation is a cost function, which most commonly is the mean square of the error between the output of the adaptive filter and the desired signal. The desired signal in this case being the remaining useful life prediction or some form of it. Similarly to the previously mentioned work by Zheng et al.[7], Miao et al. [15] uses an unscented particle filter (UPF) technique. Particle filters are known as an effective method for sequential signal tracking, however their accuracy is low, making them unsuitable for RUL

prediction. To solve this issue, Miao proposes using a UPF and applying it to a degradation model based on LI batteries. Whilst adaptive filter techniques can be useful and certainly provide accurate health prognostics they can also be prone to errors caused by variable currents and temperatures.

2) Intelligent Techniques: Intelligent techniques (IT) encompass a wide range of methods for pattern finding and data processing and as such multiple solutions appear underneath this banner. Artificial neural networks have been used notably by Zhang et al. [16], specifically using a long short-term memory (LSTM) recurrent neural network (RNN). Finally, Chen et al. [17] combines an adaptive bathtub-shaped function (ABF) with an artificial fish swarm algorithm (AFS). The ABF is utilised to model the normalised battery cycle capacity prognostic curves, before the AFS is then used to determine the optimal parameters for the ABF. These methods use a simple algorithm to predict the complex non-linear system of battery degradation and deliver more accurate results than adaptive filter techniques. Unfortunately, the methods were found wanting when analysing the uncertainty in the measurement results.

3) Stochastic Technique: Due to the complex nature of battery capacity degradation and the scenarios that batteries are often placed in, stochastic optimisation is well suited to the process of determining RUL. In 2014 Tang et al. [18] developed a method using the Weiner process with measurement error to predict RUL. The prediction of the RUL is done through truncated normal distribution, with uncertainty and drift parameters employed. Finally, maximum likelihood estimation is used to improve the estimation efficiency of the parameters. Furthermore, this model has been validated using numerous case studies, proving it to be a viable method for RUL estimation. As noted before, the stochastic process is well suited to assessing LI battery degradation, potentially more so than the previously mentioned methods. However, when the algorithm begins to consider the influence of random current and time-varying challenges, the method can struggle to produce accurate results.

4) Miscellaneous Techniques: Due to the wide ranging methods of producing the RUL there are some that don't fall under aforementioned categories, below are some of note. Using a naive Bayes (NB) based LI battery degradation model Ng et al. [19] created a model that was able to outperform SVM both in terms of accuracy and reliability. Severson et. al. [10], were able to use machine learning tools and multiple early cycles to accurately predict RUC to within 15% error on average, with the inclusion of further data reducing this error to 8%.

2.3 Research Method

The following sections will detail the theory behind the approach taken in this paper to determine RUL from a discharge curve. The previous sections cover a variety of methods to do so; however, aside from the creators of the original dataset, none were able to make use of such an extensive dataset. In the process of determining the direction of the research, the decision was made to make use of the tools available in the form of the MATLAB classification learner to perform a comprehensive look at the models available for classification of the discharge curves. Upon extracting the discharge curves from Severson et al.'s [10] data, the next step was to create identifying features that could be used for classification.

2.4 Curve Identification

In order to classify the discharge curves, values were calculated from the curves themselves in order to identify them. The decision behind creating these identifiers was to reduce computational complexity as true time series analysis can be difficult, especially given the number of discharge curves available for classification. As such the decision to represent the curves using a number of key features was chosen and proved to be effective. Figure 1 displays every 20th discharge curve for the first battery in the dataset and shows a typical voltage against time discharge graph shape for a LI battery.

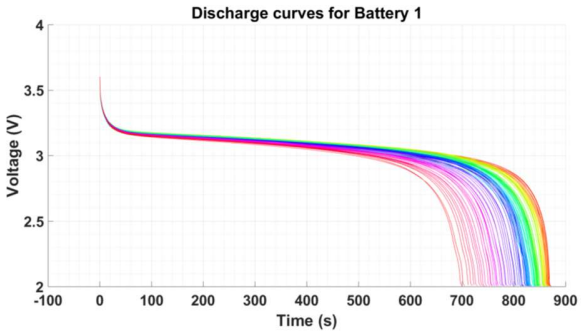

Figure 1: Every 20th discharge curve from Battery

As shown, the total discharge time varies by 150s between the shortest and longest discharges. The lower the number of previous discharges, the longer the discharge time, with this decreasing as the battery ages. This is demonstrated in the Figure by the colour gradient, which begins in red at 860s with the first discharge from the battery and increments as the cycle count increases before finally coming back to red with the last curves before failure at approximately 700s. Figure 1 is similar to all the other battery discharges from the dataset, so any comment on this specific set of curves is relevant to any battery from the dataset. The first feature was simply the time taken for a full discharge from 3.6V to 2V. Given that all batteries discharged at the same rate, this was a useful indicator of how far the battery had degraded. As is clear from Figure 1, the discharge time increases with battery age, making the total discharge time a good indicator of RUC. The second indicator was the time taken to reach the 'half-way' point of the voltage discharge. All batteries discharged from 3.6V to 2V, making the half-way mark 2.8V. This measure was used as it was felt that it was necessary to have a measure of the discharge at a 'halfway point' to contrast against the total time measure previously discussed. However, simply halving the overall time obviously produced the same distribution, therefore the relation to 'halfway point' of voltage proved a distinct feature.

Whilst using the total discharge time provided a feature with a strong correlation to RUC, it relied on a single measurement,

namely the final time measurement. In order to combat the potential for error with this feature and represent each curve as a whole, a mean value of the product of each time and voltage measurement was used as another feature. This reduced the effect that outliers had in the feature, and meant that every reading could be represented by a single figure for each curve. The equation for this feature is shown below.

$$M_j = \frac{\sum(t_i * V_i)}{N} \quad (1)$$

Where N is the number of voltage measurements taken during each discharge (the time was noted every time a voltage was taken so the number of voltage and time measurements will always be equal), M_j is simply the end result for each curve, with j indicating the individual curve, and i representing each measurement.

The next feature of the curves was influenced by the general shape of the discharge curves, as shown in Figure 1. The section of curve from roughly 100 to 500s represents a relatively constant voltage of around $3.2 - 3V$, but it can be seen that the length of this period of constant voltage is one of the main factors in the length of discharge as it is this period that varies between curves more than anything else. By subtracting the time taken to reach 3.2V from the time taken to reach 2.6V, it is possible to obtain an accurate estimation of the length of the constant voltage period. Using a wide window allowed this constant voltage period to be calculated for all the different batteries, which displayed very similar shapes but over slightly different voltage ranges. This measure of using larger ranges than needed is continued throughout to account for the different scales of the discharge curves for each battery. The next feature also takes the same window of voltage and uses the gradient as calculated according to equation 2.

$$G_i = \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i} \quad (2)$$

This is a simple calculation of the gradient between two points, and was used to work out the maximum gradient in the window from 3.2 to 2.6V for each curve. The relevant feature that is extracted from this is the time at which this maximum gradient occurs. As Figure 1 shows, the largest gradient will inevitably occur at the end of this window and provides a time value that decreases as the battery ages, giving a strong correlation across all the batteries.

The final extracted feature also uses equation 2 to calculate the gradient, and in a similar fashion calculates it over a window of voltage. In this case the window is from 3 to 2.2V and once the gradient between each point in this range is calculated, they are then averaged. As is evident from Figure 1, as the battery ages, the section of the curve in this range becomes less steep, giving another good indication of the age of the battery. This feature was chosen simply through closer observation of the curve, with the reduction in average gradient not being immediately observable in the raw data.

2.5 Matlab Classification Learner

The aim of the model is to classify the curves according to the remaining useful cycles that they represent. However, given that the longest battery cycled approximately 2250 times, this created an issue with the number of possible classes to sort the curves into. Having 2250 potential classes caused memory issues within the classification learner that on multiple occasions caused the program to crash, or caused it to 'fail' the training of the model. To solve this issue, the RUC for each curve was rounded to the nearest 10. By rounding the RUC value for each curve, the total class number was also reduced by a factor of 10, ensuring that the models no longer failed and computational times were greatly reduced. This had the consequential effect of speeding up the iteration time, allowing for more changes to be made to the final model. The MATLAB Classification learner offers a variety of potential models to train, and whilst they were predominantly used during the evaluation process this section will briefly go into detail about the various algorithms available within the program.

• Tree Classification: A classification tree is built using binary recursive partitioning, an iterative process that, during the training process, splits the data into partitions, and then on each branch further partitions the data. During prediction the data is fed into the tree and is appropriately partitioned before finally being assigned a class.
• Discriminant Analysis: During discriminant analysis it is assumed that different classes create data based on different Gaussian distributions. Discriminant analysis is used to reduce the dimensionality of a dataset and is often used as a pre-processing step; however, as is shown, it is possible to use it for multi-class classification.
• Support Vector Machine (SVM): The goal of an SVM algorithm is to create hyperplanes in an N-dimensional space (where N is the number of features) that distinctly classifies the data points. This means multiple hyperplanes are to be created to classify the data, and as such, each hyperplane is chosen on the maximum margin it leaves between data points, with larger margins being preferable.
• K Nearest Neighbours was the algorithm chosen for further work as it was the most accurate, and as such, it is discussed in much finer detail in the next section.

2.6 Weighted K Nearest Neighbours

K nearest neighbour (k-NN) classification is the algorithm that is used for the prediction of remaining useful cycles. It was chosen as the best option using the MATLAB Classification Learner; however, this process will be discussed in future sections. k-NN classification is widely considered to be one of the most fundamental and simple classification methods and is often used as the first algorithm when there is little knowledge of the distribution of the data. Its relative simplicity allows for less computation and faster run times, making it a popular choice. It was created as a way to perform discriminant analysis when the parametric estimates of probability densities are either unreliable, unknown, or difficult to determine. k-NN is a

'supervised' classification method due to the fact that it uses the class labels of the training data. This is opposed to 'unsupervised' methods, which employ clustering and do not use the class labels of the training data. In order to classify the data, the algorithm first splits the samples into training and test sample categories, in this case x_i and x respectively. From here the classes for the training and test samples can be established with ω being the true class of the training sample and $\widehat{\omega}$ indicating the predicted class for the test sample.

$$\widehat{\omega},\ \omega = 1.2, \dots, \Omega \tag{3}$$

where **Ω** is the total number of classes. During the training phase only the true class of ω for each training sample is used; however, when testing, the $\widehat{\omega}$ is predicted. The 'k' in k-NN denotes the number of neighbours that are used in the classification of the sample. For example, when k = 1 the predicted class ($\widehat{\omega}$) of the test sample x is set equal to the true class ω of its nearest neighbour, as shown below.

$$\widehat{\omega} = min.dist.(\omega) \tag{4}$$

When k increases above 1 the predicted class of the test sample is set to the most frequent true class among the k nearest training samples. This gives the decision rule,

$$D : x \rightarrow \widehat{\omega} \tag{5}$$

With each x, the test sample being predicted is the calculated $\widehat{\omega}$. In order to establish the 'nearest neighbour' to each point there are a variety of possible distance metrics that can be used. The classification learner offers many options for the distance metric when optimising the model. Given that these are all different in their calculation, this paper briefly covers the final choice, the City Block distance metric. This method is a special case of the 'Minkowski' metric [20], the formula for which is shown in equation 6,

$$D\ (x, y) = \sqrt[p]{\Sigma_d |x_d - y_d| p} \tag{6}$$

In the case of the City Block metric, p = 1, making the calculation for distance using the City Block method simply the sum of the absolute difference of the Cartesian coordinates. The term 'weighted' stems from a slight variation to the normal k-NN algorithm. Upon determining the 'k' nearest neighbours the algorithm then weights each point according to its distance from the point to be predicted. This weighting often takes the form of an inverse or squared inverse rule, as shown below.

$$W_i = \frac{1}{di}\ or\ \frac{1}{di^2} \tag{7}$$

With W being the weight assigned to each point, and d representing its calculated distance from the predicted point. From here the total weight of each class is summed and the highest weighted class within the 'k' nearest neighbours is predicted as $\widehat{\omega}$. Weighting the predictions can mean that within the subset of 'k' nearest neighbours it is no longer the most frequent class that is chosen, but the class with the highest weight.

In order to improve the accuracy of the predictions the input features can be transformed prior to analysis. Given the difference in average values between the features (as displayed in Table I) it is clear that the input would benefit from standardisation. This is the process of removing scale effects caused by different measurement scales, or in the case of this paper reducing the large difference in values of the curve identifiers for the discharge curves. The process of standardization transforms the initial feature values into *z-scores* using the values of mean and standard deviation over all input samples, as given below

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{8}$$

In this relationship x_{ij} is the value for the ith sample and the jth feature, μ_j is the average of all x_{ij} for feature j, and σ_j is the standard deviation of all x_{ij} over all input samples. Once the standardization has taken place the range and scale of the z-scores should be similar, providing that the distributions of the input features were also similar.

The final aspect of creating the algorithm in MATLAB was to use cross-validation when assessing the performance. Generally speaking, if predictions are made using any of the training sets, it will be expected that the algorithm will be more likely to correctly predict these classes and as such this will result in false increases in accuracy when evaluated. In order to ensure any evaluation of accuracy was reliable, cross-validation was used. In the method followed by this report, 5-fold validation was used within the MATLAB classification learner, which meant splitting the data into five equally sized partitions. From here, partitions 1 to 4 are used for training while partition 5 is used for testing. This is repeated as partitions 1 to 3 and 5 are now used for training and partition 4 is used for testing. Eventually all partitions will be used to test, ensuring the accuracy given is as reliable as possible.

3. METHOD

3.1 Initial Data Collection

Commercially available high-power LFP/graphite A123 APR18650M1A cells were used to create the dataset, the cells having a nominal capacity of 1.1Ah and a nominal voltage of 3.3V [21]. The cells were cycled in cylindrical fixtures with 4-point contacts on a 48-channel Arbin LBT battery testing cycler. Throughout the tests, the temperature was kept at a steady 30◦C using an environmental chamber. In order to create differing conditions, the cells were cycled with various fast charging policies. Cells were charged from 0% to 80% state of charge (SOC) using various single-step and two-step charging policies. The batteries exhibited charging times in the region of 540 to 800s during this period. From 80% to 100% SOC the batteries were uniformly charged at 1C (Constant Current - Constant Voltage) CC-CV charging step to 3.6V. From here all cells were discharged using CC-CV at 4C down to 2.0V. These voltage cut-offs were recommended by the manufacturer. The capacity of

each battery was calculated as a function of voltage and evaluated for each cycle, and the batteries were stopped cycling when the capacity reached 80% of the original capacity or 0.88Ah. This was chosen as the cut-off point or point of 'failure' as this is a regularly used industry value for establishing when a battery is no longer fit for purpose. This paper notes that this work is unaffected by the decision to make 80% the actual 'failure' point as this number is arbitrary. It simply limits the model to only being able to predict remaining useful cycles in the region of 80-100%. In the process of writing the code, steps were taken to ensure that the code was written in such a way as to make altering the model to predict remaining useful cycles in a larger region as easy as possible. It is unfortunately impossible to predict remaining useful cycles outside of this range as the data used didn't supply discharge curves beyond when the battery reached 80% capacity. Should such data become available it would be simple to retrain the model and allow it to predict the remaining useful cycles until capacity reaches anywhere in the region the data provides.

3.2 Data Processing

Once the data had been created it was necessary to thoroughly process it in order for it to be compatible with MATLAB's classification learner [11]. Given that all the discharges were over the same voltage range, it was possible to apply the code and filters identically across all the batteries. As mentioned previously, each battery was charged under differing policies meaning that it was unsuitable to use the charge curves when training the algorithm as it would create too much variability.

The first operation was to separate the discharge curves from the rest of the cycle as the dataset didn't separate them. The code in Figure 2 shows the process of removing all the charge readings by finding the first point at which the data begins to drop below 3.6V, indicating the beginning of the discharging process. The actual value used to determine when the discharge started within the code was 3.5995V, the last value greater than 3.5995V being counted as the first value of the discharge curve. This measure was due to the battery being left at 3.6V for a period of time meaning that the discharge curve began with a flat line when using 3.6V as the cut-off for the beginning of the curve. It was through trial and error that 3.5995V was found to reliably give the correct start point of the discharge curve.

As shown in the code, the curve was then cut to ensure that it finished at 2V. This was due to the testing sometimes having been extended into the next charging cycle and as such leaving incorrect curves. Finally, each curve was evaluated to ensure it fell within the expected time parameters. At this point the time data was still in minutes, therefore anything that took over 100 minutes to discharge was discarded as an outlier curve and set to zero. From here they were no longer included in the calculation and as such could not incorrectly influence the training of the algorithm.

```
%discurves is the variable that contains the original data
%check to find where the battery begins to discharge
Chk = discurves(a).cycles(b).V >= 3.5995;
Index1 = find(chk);
Vstart = max (index1);
%include all data from start of discharge
for i=1: length(discurves(a).cycles(b).V) – Vstart
allV1(i) = [discurves(a).cycles(b).V(Vstart)];
allt1(i) = [discurves(a).cycles(b).t(Vstart)];
Vstart = Vstart +1;
end
chk = allV1<2;
index1 = find(chk);
Vend = min(index1);
  %remove any excess data after V=2
for i=1:Vend
     allV2(i) = allV1(i);
%set t=0 for the beginning of discharge
allt2(i) = allt1(i)-allt1(1);
end
discurves(a).cycles(b).V =allV2';
discurves(a).cycles(b).t = allt2';
clearvars – except batch_combined discurves a b;
%check for whole curves that were incorrectly recorded and remove these
tf3 = (100 < discurves (a). cycles(b).t);
tf3 = mean (tf3);
if tf3>0
   discurves (a).cycles(b).V =zeros;
   discurves(a).cycles(b).t =zeros;
end
clear tf3;
```

Figure 2: Discharge curve separation

When inspecting the curves after the first round of processing it was noted that, due to measurement faults, some of the curves had very few readings in comparison to the vast majority. Similarly to the previously mentioned curves that took too long to discharge, these curves were set to zero and ignored in any further work. It was felt that replicating these curves would be contradicting the final aim of using single curves as an input into the algorithm, hence why the outlier curves were removed as opposed to filling the data by using extrapolation from previous and subsequent curves. From here the time data was converted to seconds. Whilst this made the values much larger as they had previously been in minutes, having all data in SI units was deemed to be more important, and it was noted that the standardisation of values that was discussed earlier would render this the correct choice. Finally, each curve was filtered using a Savitzky-Golay (SG) filter in order to smooth the data and remove any unwanted noise. Given that experimental data is subject to noise, this greatly increased accuracy of any predictions. The SG filter is a digital filter that can be applied to increase the precision of the data without distorting the trend of the discharge curve in the first place. The filter uses convolution to fit sub-sets of adjacent data points with a low-degree polynomial by the method of linear least squares, giving a smoother discharge curve with less noise. This filtering would eventually be applied to each input curve when making predictions.

3.3 Feature Extraction

Following on from the processing of the discharge curves, it was necessary to extract the features discussed earlier in order to train the model. This section describes the features that were chosen to represent each curve when classifying. Those were

chosen as part of a testing process that involved plotting the feature against the remaining cycles to determine the amount of correlation between the two. Features that exhibited higher correlation were then chosen for training the model. Figure 3 displays the relationship between one of the features and the remaining useful cycles. As is clear from the graph when the remaining useful cycles are lower, e.g. in the region of 0-500 remaining cycles, the relationship is much stronger with a small change in remaining cycles, exhibiting a much greater change in maximum gradient in comparison to earlier cycles. This relationship between RUC and the trends of the features will be discussed in later sections.

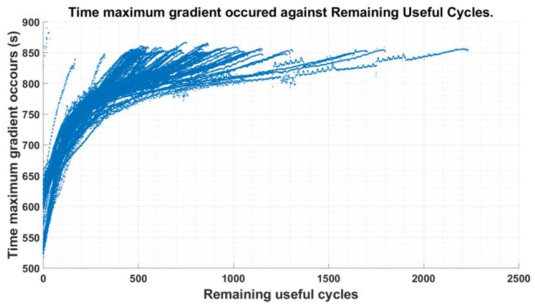

Figure 3: Maximum gradient against remaining useful cycles

Figure 4 displays the first two processes to extract the data. Line 3 of the code in Figure 4 displays the simple mean calculation. Given the large number of curves, wherever possible it was decided that the feature extraction should be computationally simple in order to reduce processing time, even so the program still takes upwards of 10 minutes to run. The next process was to create the two 'windows' of voltage that were used when extracting features. In this case the windows spanned from 3.2 to 2.6V and from 3.0 to 2.2V. Figure 5 shows the creation of the 3.2 to 2.6V window. When creating this window the code uses the number of the voltage measurement to indicate the start and end of the window, shown in Figure 5 as Vhf1 and Vhf2. For example, if the voltage drops below 3.2V on the 100th measurement of voltage Vhf1 would equal 100. The creation of this voltage window is identical to the creation of the 3.0 to 2.2V window, hence why it is not included. Figure 5 also displays the creation of the feature (Tdiff) displaying the period of time that the curve remained at a 'constant' voltage on line 17.

```
%calculation of the mean of the product of V and t coordinates
Tmn = mean (discuves(m).cycles(i).t.* discurves(m).cycles(i). V);

%establishing total discharge time
Tmax = max(discurves(m).cycles(i).t);

%establishing the reading whereby the voltage reached 2.8V
chk = discurves(m).cycles(i).V <= 2.800;
index1-find(chk);
Vhf = min(index1);
clear chk index1;
```

Figure 4 Process to extract battery data

```
%creating the window from 3.2V to 2.6V
chk = discurves(m).cycles(i).V <= 3.200 ;
index1 = find (chk);
Vhf1 = min(index1);
Tstart(i) = discurves(m).cycles(i).t(Vhf1);

clear chk index1;

chk = discurves(m).cycles(i).V <=2.600;
index1 = find(chk);
Vhf2 = min(index1);
Tstart2(i) = discurves(m).cycles(i).t(Vhf2);

clear chk index1;

Tdiff = Tstart2(i) – Tstart(i);
```

Figure 5 Creation of voltage measurement windows

The final aspect to the feature extraction was to acquire the gradient between subsequent points on each discharge curve, the process for which is shown in Figure 6. The 'for loop' in Figure 6 shows the process for calculating the gradient between two points. On the occasion that the voltage had been recorded as the same number twice, the loop simply takes the next point to calculate the gradient from in order to avoid a zero value. Appendix A-E displays the 'for loop' used to calculate the position of the maximum gradient, using a simple increment system that evaluates the most recent gradient calculation against the highest value before replacing if the new value is higher. As mentioned previously the process for extracting the mean gradient in the 3.0 to 2.2V window was almost identical. Finally, the data was placed in a 'Table' type variable, as shown in Table 1, as this was the format the classification learner required.

```
%calculating the maximum gradient
for x=Vhf1:Vhf2
    x1=discurves(m).cycles(i).t(x);
    x2=discurves(m).cycles(i).t(x+1);
    y1=discurves(m).cycles(i).V(x);
    y2=discurves(m).cycles(i).V(x+1);

    if y2==y1
        y2 = discurves(m).cycles(i).V(x+2);
        x2 = discurves(m).cycles(i).t(x+2)
        end
        dydxcnt = (y2-y1)/(x2-x1);
        if dydxcnt <=dydxmax
              dydxmax = x;
        end
end
```

Figure 6 Feature extraction to acquire the gradient between subsequent points on each discharge curve

Table 1: Example input table showing the first 3 rows.

RUC	dydx Max	t Max	Vhf	t Diff	t Mean	Dydx Mean
1850	845.7	864.9	817.3	804.8	1326.5	-0.0049
1850	846.2	865.9	817.8	805.2	1329.2	-0.0047
1850	846.6	866.6	818.1	805.6	1331.2	-0.0046
…	…	…	…	…	…	…

The headings of the table are as follows:

- 'RUC' is the remaining useful cycles before the battery reaches 80% capacity
- 'dydx Max' is the time in seconds at which the maximum gradient occurs in the window of 3.2 to 2.6V

• ’t Max’ is the total discharge time in seconds
• ’Vhf’ is the time at which the voltage reaches 2.8V or half-way through discharging in seconds
• ’t Diff.’ is the time period in seconds between the voltage reaching 3.2 and 2.6V
• ’t Mean’ is the mean of the product of each time and voltage coordinate
• ’dydx Mean’ is the mean gradient between 3.0 to 2.2V

3.4 Model Creation and Optimisation

Once each curve had the features extracted and appropriately formatted it was possible to use MATLAB’s classification learner (MCL) [11] to create and optimise the model. Before its input to the MCL, the data was split on a 7 : 3 ratio to create Training and Testing inputs, to ensure that any accuracy claims were as valid as possible and that the model was able to train on separate data. From here all models were trained using the same ’Training’ dataset, with the ’Testing’ dataset being used exclusively for evaluation. The whole process of model optimisation was somewhat iterative in that any time a new feature was created or considered, it would be tested using the MCL, specifically its ’Quick to Train’ option. This option meant that it was possible to train the input data on a variety of k-NN and decision tree algorithms that were computationally inexpensive and gave quick results, indicating the effect of the new feature or alteration to the data processing on the model accuracy. Whilst this presented a limited view of the other potential models that could be trained, through testing it was clear that any changes on the accuracy of these models were indicative of similar changes to any of the models that took longer to train. This meant that the process of improvement was quick and made it possible to make numerous changes without losing large amounts of time to re-train the models to see the changes in accuracy. It should be noted that the ’accuracy’ metric within the MCL is fairly simplistic, simply indicating the overall number of correct predictions as a percentage of the total predictions. Whilst this doesn’t offer an overall view of model performance, throughout the course of experimentation it was validated to provide a satisfactory overview of performance that was indicative of the model performance in other metrics, hence why it was confidently used throughout as a judge of a model’s accuracy. Once the features had been finalised through testing, all models that the MCL offered were trained on the data to establish which gave the greatest accuracy. Once weighted k-NN had been arrived at as giving the greatest accuracy, the parameters of the k-NN model were altered using the MCL’s built in tools. In order to establish the best possible accuracy, the parameters were altered in the following order:

1) Principal Component Analysis (PCA). PCA is a procedure that uses transformation of the observations into a set of values that are linearly uncorrelated, called principal components. MCL allows for this to be easily used, but given the large reduction in accuracy when turned on, it was decided that it would be not be employed.

2) Standardisation. As discussed earlier, this was a key factor in improving accuracy due to the wide range of feature values, demonstrated in Table 1. Standardisation was used throughout the refining of the k-NN model.
3) Distance Weight. This value changed little when varying the choice between ’Inverse’ and ’Squared Inverse’, however there was a noticeable drop in accuracy when using ’Equal’ weighting as this meant that the k-NN was no longer ’Weighted’ in any way.
4) Distance metric. As discussed earlier this had some effect but after some experimentation it was observed that the ’City Block’ metric was the most accurate.
5) ’k’ Value. The lower the ’k’ value the more likely the prediction is to be correct, however it is more susceptible to outliers, therefore it is important to find a balance.

These factors were altered one at a time from the default settings of ’Weighted KNN’ in the MCL. The default settings were: k = 10, the distance metric was Euclidean, the distance weighting was ’Squared Inverse’, and PCA was turned off. From here the model was exported allowing it to be used for further predictions and testing for accuracy.

4. RESULTS AND DISCUSSION

4.1 Machine Learning Results

The first investigation using the MCL was to see which model produced the most accurate results. During testing only the ’Quick to Train’ models were used, however once the input features had been finalised all the models were trained. The predominant method of evaluating the performance of the models will be using the ’Accuracy’ value provided by the MCL, which is shown as

$$Acc. = \frac{Correct\ predictions}{Total\ predictions} \quad (9)$$

Within the MCL, the models were grouped into categories. For example, both ’Weighted k-NN’ and ’Fine k-NN’ fell under the ’k-NN’ denomination. As such, when evaluating performance this report only includes the most accurate from each category of model or, in the case of the ’k-NN’ category, the three most accurate models as a point of comparison. Table 2 displays the model categories, with the most accurate specific model in brackets, as well as the accuracy for each model.

Table 2: Accuracies of tested models.

Model	**Accuracy (%)**
Tree (Fine)	10.9
Discriminant (Linear*)	6.5
SVM (Linear*)	12.4
Fine KNN	25.4
Weighted KNN	26.2
Medium KNN	22.1

As is clear from Table 2 there are some models that were far better suited to the task of classifying the curves. ’*’ indicates where other models in the category failed and it was impossible to test them. This paper did not have access to hardware that

might have been able to handle the complexity of these failed models. Whilst this is regrettable it is believed that the low accuracies in the less complex tests are indicative of the accuracies these more complex models would have achieved and therefore doesn't affect the final outcome.

4.2 Final Algorithm Optimisation

The next process was to optimise the Weighted k-NN model to ensure that it was producing as accurate predictions as possible. Table 3 displays the effects of changing the model variables, as discussed in Section 3.4. Whilst some of the effects were small, it built confidence in the model as a whole.

As is clear, altering the variables had little effect when increasing the accuracy of the model, indicating perhaps that the majority of the accuracy came from initial model choice and the input features used for classification. Given that the use of the City Block metric was the only variable to increase accuracy, it remained in the final model as a chosen metric.

Table 3: Effects of variable changes on model accuracy.

Variable Altered	Value	Accuracy (%)
'k' Value	1	25.4
	2	25.4
	3	25.7
	5	26.0
	10*	26.2
	20	26.1
	50	25.7
PCA	OFF*	26.2
	ON	19.8
Distance Weight	Equal	10.1
	Inverse	25.3
	Squared Inverse*	26.2
	Euclidean	26.2
	Chebychev	23.0
	Mahalanobis	18.5
	Cosine	14.3
	City Block	29.1

'*' Indicates where this value was the default for k-NN

4.3 Final Algorithm Accuracy

The model's accuracy was somewhat difficult to categorise due to the large number of classes. To give an example of why this was an issue, when working out classification accuracy, if the model was out by 1 class or 10 cycles this would be counted as an incorrect prediction. However, if the total cycle life for this battery was 1000 cycles this could mean the prediction was only incorrect by approximately 1%. As such, while classification accuracy has been included in the final figures it should not be regarded as being completely representative of the accuracy of the model. Table 4 displays various accuracy metrics to give a comprehensive overlook of the model's performance.

Table 4: Final model accuracy

Accuracy Test	Accuracy
Classification Accuracy*	7.844%
Classification Accuracy	0.8310%
Mean Percentage Error*	14.64%
Mean Percentage Error	14.64%
Mean Relative Difference*	21.73%
Mean Relative Difference	21.95%
Adjusted R Squared*	0.5200
Adjusted R Squared	0.5200

'*' Indicates where the actual value used in the accuracy calculation is rounded to the nearest 10, as all the class predictions are to the nearest 10. Percentage error was calculated as,

$$Mean\%Error = \frac{[Actual\ RCL - Predicte\ \ RCL]}{Total\ Battery\ Cycles} * 100 \qquad (10)$$

Relative Difference was calculated as,

$$MeanRel.Diff. = \frac{|Actual\ RCL - Predicte\ \ RCL|}{\max(Actual\ RCL, Predicted\ RCL)} * 100 \qquad (11)$$

As is clear, the classification accuracy is low, however due to the high number of classes this was to be expected. The other metrics for determining accuracy, however, create much more confidence in the accuracy of the predictions. It is clear that the k-NN model could be feasibly used for rapid consumer testing of the RUC of a battery, simply by obtaining one discharge curve. The use of a single discharge curve as opposed to many, while still maintaining a similar accuracy, is an important step in the classification of RUC. Particularly in comparison to work done by Severson et. al. [10] it can be seen that the average error of this model is almost identical when using a single curve.

4.4 Use Case and User Interface

As mentioned above, the use case for this model would be within the consumer environment, whereby a user might require a fast and relatively accurate estimate of the number of cycles left on their battery. As part of proving this use case, a workable UI was created to demonstrate this, a screen-shot of which can be seen in Figure 7. The viability of this UI was proved in its simple interface that was tested among numerous peers who did not have a scientific background as a potential use case. In all cases it was reported that the UI was intuitive and provided the information required. Appendix F displays the UI during a potential user interaction. The graph has been toggled to currently display the raw input curves that were fed to the model before any refining. The table on the left hand side of the screen-shot displays the results from a random selection of 25 curves from the training dataset, hence why the actual RUL is known and the percentage error is able to be calculated. If a user were to input their own curves the table would simply display the predictions of RUL for each curve.

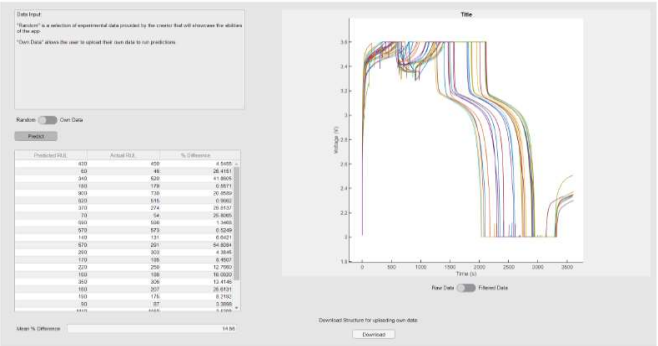

Figure 7 Battery testing user interface

5. CONCLUSIONS

This paper investigates the viability of using classification algorithms for the identification of remaining useful cycles of a Lithium-Ion battery from a singular discharge curve. Using the built in features within the MATLAB platform, the large dataset from Severson et al. [10] was processed and the discharge curves were extracted from the cycle data. From here the data was filtered before six features were extracted from each discharge curve and used to classify the remaining cycle life of the battery. Using MATLAB's built in classification learner, various classification models were trained using the extracted data and evaluated against each other using the accuracy value provided by the learner. After evaluation, Weighted k Nearest Neighbours emerged as the most consistently accurate model and was chosen for further improvement. The parameters of the k-NN model were then altered in order to determine the most accurate combination of parameters. After this modification the model was extracted for both further accuracy testing and the creation of a user interface for demonstration of the viability of the model as a commercial and private tool. The final model accurately classified discharge curves 7.84% of the time, but given the large number of potential classes the other indicators of model accuracy, such as a mean percentage error of 14.6%, gave a greater indication of the potential and accuracy that this model has. Further investigations into classifying the discharge curves when taking into account multiple consecutive discharges reduced this percentage error to 12.1%. The UI was created using MATLAB's app designer and ensured an intuitive demonstration of the usefulness and accuracy of this model.

REFERENCES

[1] N. N. et. al., "Li-ion battery materials: present and future," Materials Today, vol. 18, no. 5, pp. 252 – 264, June 2015.

[2] I. W. et al. (2019) Lithium-ion batteries - statistics and facts. [Online]. Available: https://www.statista.com/topics/2049/lithium-ionbattery-industry/

[3] A. C. et al., "Advancement in battery technology: A state-of-the-art review." in 2016 IEEE Industry Applications Society Annual Meeting, 2016, pp. 1–10.

[4] M. L. et al., "30 years of lithium-ion batteries." Advanced Materials, vol. 30, 2018.

[5] A123Systems. (2009) High power lithium ion apr18650m1a. [Online]. Available: https://www.batteryspace.com/prod-specs/6612.pdf

[6] H. X. et al., "A review on the key issues of the lithium ion battery degradation among the whole life cycle." eTransportation, vol. 1, August 2019.

[7] Z. X. et al., "An integrated unscented kalman filter and relevance vector regression approach for lithium-ion battery remaining useful life and short-term capacity prediction." Reliability Engineering & System Safety, vol. 144, pp. 74–82, Dec 2015.

[8] W. S. et al., "Prognostics of lithium-ion batteries based on battery performance analysis and flexible support vector regression." Energies, vol. 67, pp. 6492–6508, October 2014.

[9] W. D. et al., "Prognostics of lithium-ion batteries based on relevance vectors and a conditional three-parameter capacity degradation model." Journal of Power Sources, vol. 239, 2013.

[10] S. K. et al., "Data-driven prediction of battery cycle life before capacity degradation," Nature Energy, vol. 4, p. 383–391, 2019 2019.

[11] MATLAB. (2020) Classification learner. [Online]. Available: https://uk.mathworks.com/help/stats/classificationlearner-app.html

[12] B. I, Batteries in a Portable World, 4th ed., 2016.

[13] X. Y. et al., "Battery management systems in electric and hybrid vehicles," Energies, December 2011.

[14] H. A. M. et al., "A review of state of health and remaining useful life estimation methods for lithium-ion battery in electric vehicles: Challenges and recommendations." Journal of Cleaner Production, vol. 205, pp. 115–133, 2018.

[15] M. Q. et al., "Remaining useful life prediction of lithium-ion battery with unscented particle filter technique." Microelectronics Reliability, vol. 53, pp. 805–810, June 2013.

[16] Y. Z. et al., "Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries." IEEE Transactions on Vehicular Technology, vol. 67, pp. 5695–5705, July 2018.

[17] C. L. et al., "Quantitative analysis of lithium-ion battery capacity prediction via adaptive bathtub-shaped function." Energies, vol. 6, pp. 3082–3096, 2013.

[18] T. S. et al., "Remaining useful life prediction of lithium-ion batteries based on the wiener process with measurement error." Energies, vol. 7, pp. 520–547, 2013.

[19] N. S. et al., "A naive bayes model for robust remaining useful life prediction of lithium-ion battery." Applied Energy, vol. 118, pp. 114– 123, April 2014.

[20] C. et al., "An empirical study of distance metrics for k-nearest neighbor algorithm," in 3rd International Conference on Industrial Application Engineering, Jan 2015, pp. 280–285.

[21] S. K. et al. (2019) Data-driven prediction of battery cycle life before capacity degradation. [Online]. Available: https://data.matr.io/1/projects/5c48dd2bc625d700019f3204

[22] MATLAB. (2020) Matlab app designer. [Online]. Available: https://uk.mathworks.com/products/matlab/app-designer.html