ChE 492

PROJECT

UNDERSTANDING SUPERCAPACITOR CHARGE/DISCHARGE RATES USING MICROSTRUCTURE IMAGE PROCESSING

by

Burak Can KOBAN

Course Instructor	: Asst. Prof. Betül URALCAN
Advisor	: Asst. Prof. Betül URALCAN
Date of Submission	: 07.07.2021

Department of Chemical Engineering

Boğaziçi University

Bebek, Istanbul

ABSTRACT

The project aims to design machine learning algorithm which is able to predict energy densities of supercapacitors by using input data that is enriched from the results came from image processing techniques. The created machine learning algorithm will make possible to save time and money spent during material selection process of supercapacitors. Input data has extent of 22 features and 2189 observations. By using image processing, porosity and pore volume columns are improved since 84 of the 628 predicted data is filled with more the results come from more advanced technique. Enriched data is used to train 5 different machine learning models that are namely linear regression, ridge regression, lasso regression, regression tree and artificial neural network. The RMSE test results of the trained models are 5.2343, 5.2335, 5.1288, 3.7315 and 5.010, relatively. The results are satisfactory which implies that models predict energy densities accurately.

TABLE OF CONTENTS

ABSTRACT **1**

TABLE OF CONTENTS **2**

LIST OF TABLES **4**

LIST OF FIGURES **5**

1. INTRODUCTION **6**

2. THEORY **7**

2.1. Supercapacitors 7

2.2. Correlation Matrix 8

2.3. F-Test 8

2.4. K-Fold Cross Validation 9

2.5. Linear Regression 9

2.6. Ridge Regression 11

2.7. Lasso Regression 11

2.8. Regression Tree 12

2.9. Neural Network 13

2.10. Errors: R^2 and RMSE 14

3. PRE-PROCESSING **14**

3.1. SEM Image Data Collection 15

3.1.1. Algorithm and Results 16

3.2. Filtering Data 18

3.3. Normalization & Dummy Coding 20

4. FEATURE SELECTION **21**

4.1. Correlation Matrix and Heat Map 21

4.2. Filter Type Feature Selection: F-Test 23

4.3. Trial and Error Method 24

5. PARAMETER TUNING & MODEL TRAINING **26**

6. RESULTS AND DISCUSSION 27

6.1. Linear Regression 27

6.2. Ridge Regression 28

6.3. Lasso Regression 29

6.4. Regression Tree 30

6.5. Artificial Neural Network 31

7. CONCLUSION 35

8. FUTURE THOUGHTS 36

9. REFERENCES 37

APPENDIX 39

Image Processing 39

Correlation Matrix 40

Model Training & Predictions 42

LIST OF TABLES

Table 3.1 Features and response variable list 15
Table 3.2 Filtering data according to RMSE values 19
Table 3.3 Representative configuration for dummy coding 21
Table 4.1 RMSE Results According to Elimination of Features 24
Table 4.2 Percentage Differences in RMSE for Each Eliminated Feature 25
Table 5.1 Data splitting names and percentages 26
Table 5.2 Tuned models and parameters of the models 27
Table 6.1 Error values for linear regression 27
Table 6.2 Error values for linear and ridge regression 28
Table 6.3 Error values for linear, ridge and lasso regression 29
Table 6.4 Error values for linear, ridge and lasso regression and regression tree 30
Table 6.5 Error values for linear, ridge and lasso regression, regression tree and ANN 31

LIST OF FIGURES

Figure 2.1 Representation of supercapacitor 7
Figure 2.2 5-Fold Cross Validation Representation (Patro, 2021) 9
Figure 2.3 Best Linear Regression Model Representation and Residuals 10
Figure 2.4 Typical Neural Network Representation (IBM , 2020) 13
Figure 3.1 SEM image processing results a) Original image b) Depth map visualization c) Pore space segmentation d) Binary segmentation 17
Figure 3.2 Actual values versus predicted values without any filtering data 18
Figure 3.3 RMSE result according to percentage of data maintained 19
Figure 3.4 The predicted values versus actual values graph for 70 % of the data maintained 20
Figure 4.1 Correlation matrix results and heat map of the features 22
Figure 4.2 Rank of the features according to predictor importance results 23
Figure 6.1 Prediction values versus actual values for linear regression 28
Figure 6.2 Prediction values versus actual values for ridge regression 29
Figure 6.3 Prediction values versus actual values for lasso regression 30
Figure 6.4 Prediction values versus actual values for linear regression 31
Figure 6.5 Hidden layers representation for ANN 32
Figure 6.6 Mean squared error changes according to iterations 32
Figure 6.8 Error histogram of training, validation, and test data 33
Figure 6.9 Actual values versus predicted values for ANN from test data 33
Figure 6.10 Actual values versus predicted values for ANN from validation data 34
Figure 6.11 Actual values versus predicted values for ANN from training data 34
Figure 6.12 Actual values versus predicted values for ANN from all data 35

1. INTRODUCTION

Supercapacitors are new generation energy storage devices. Supercapacitors are open to improvement and have research area since they are new technologies compared to batteries and fuel cells. Fundamentally, supercapacitors are able to substitute batteries successfully in many areas where batteries remain insufficient in many aspects (Gu & Yushin, 2013). For example, supercapacitors are very good at applications where a great number of charge/discharge rates and long-life period are necessary (Simon & Gogotsi , 2008). Therefore, increasing applications of supercapacitors in the fields, where fast charge/discharge rates and long-life periods are required, is only possible with enhancing energy densities of supercapacitors.

Energy densities and other performance parameters such as capacitance and power densities are highly dependent on material properties of supercapacitors. Therefore, material selection process is very important step during supercapacitor design. However, since material selection process is handled by using trial and error experiments in real life, material selection is very time and money consuming process for researchers. The problem can be overcome by using accumulated data which is created from the supercapacitor experiments in literature to train machine learning models and form a model for predicting performance parameters of supercapacitors. Similar applications are made in the field of batteries where performance parameters of batteries are predicted by using machine learning models (Deringer, 2020) . However, machine learning applications in supercapacitors are deficient in the literature.

In this project, it is aimed to obtain machine learning models where performance parameters of supercapacitors can be predicted. To achieve this goal, comprehensive and stratified investigation is conducted. To train the supercapacitor model, data is provided by Asst. Prof. Betul Uralcan and members of her laboratory called Soft Matter Laboratory which is located in Bogazici University. The missing porosity and pore volume values in the provided data were filled with averaging available data. The missing porosity and pore volume cells is filled with results by provided SEM image processing technique which is enriched the training, testing and validation data for the model. Before the model is created, the data is filtered to remove outliers and three feature selection method is applied to eliminate less informative data. The feature selection methods are namely correlation matrix, F-test and trail & error method. Finally, 5 different models are trained to predict energy density of supercapacitors as a response variable. The machine learning models used in this project are linear regression, lasso regression ridge regression, regression tree and artificial neural network. The results are also provided for each model and discussed with details.

2. THEORY

2.1. Supercapacitors

The working principle of supercapacitors are very similar to other energy storage device. By ensuring accumulation of electrons on one of the electrodes of the supercapacitor, potential difference between the electrodes is created which provides energy storage to supercapacitor (Conway, 1999). The unique property of supercapacitors during energy storage is keeping ions on the high surface area electrodes with adsorption (Salanne, et al., 2016). Storing ions with adsorption distinguish supercapacitors from batteries because batteries are bounded with the limits of diffusion rules. Therefore, adsorption remove this barrier as a result high energy density required situations can be handled (Miller & Simon, 2008).

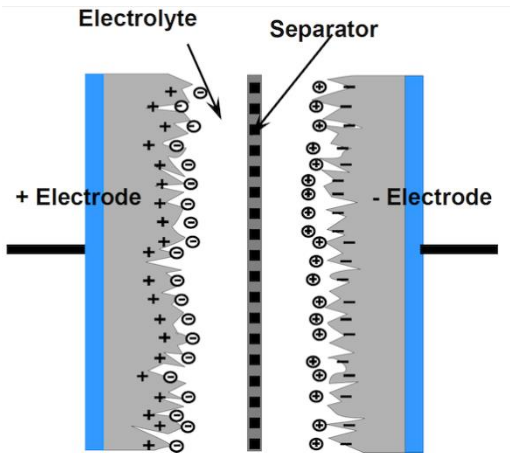

Figure 2.1 Representation of supercapacitor

In this project, energy density is selected as dependent variable. Energy density is the energy stored in the one unit of the volume. In the following formula, the relationship between capacitance and energy density is formulated (Serway & Jewett, 2013).

$$E = \frac{CV^2}{2} \tag{2.1}$$

where,

E is energy density,

C is capacitance,

V is potential difference.

Capacitance is amount of change in the electrical charge when the change in the electrical potential is occurred (Serway & Jewett, 2013).

$$C = \frac{q}{V} \tag{2.2}$$

where q is the electrical charge.

2.2. Correlation Matrix

When working with two or more random variables, it is valuable to understand how two variables vary together (Montgomery & Runger, 2014). It is important to define relationship between features. To do that, creating correlation matrix gives good insight about the variable relationships. The following equation clarifies the correlation between two variables (Montgomery & Runger, 2014).

$$\rho_{XY} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{2.3}$$

where

ρ_{XY} is the correlation coefficient between the variables x and y,

x_i is the values of the x variable in a sample,

$\bar{x}$ is the mean of the values in the x variable,

y_i is the values of the y variable in a sample,

$\bar{y}$ is the mean of the values in the y variable.

Correlation between any two variables have to be in the range of -1 to 1.

$$-1 \leq \rho_{XY} \leq 1 \tag{2.4}$$

After calculating every correlation coefficient between two variables in the extent of n variables, n by n correlation matrix can be formed.

2.3. F-Test

X_{11}, X_{12}, …, X_{1n1} and X_{21}, X_{22}, …, X_{2n2} are random samples from normal distributions with standard deviations and means of σ_1^2, σ_2^2, μ_1 and μ_2, relatively. Both populations are assumed to be independent. Lastly, S_1^2 and S_2^2 are the sample variances of random samples. Then the following ratio has an F distribution with n_1-1 degrees of freedom in numerator and n_2 -1 degrees of freedom in denominator (Montgomery & Runger, 2014).

$$F = \frac{S_1^2/\sigma_1^2}{S_2^2\ /\sigma_2^2} \tag{2.5}$$

The next step is evaluation of null hypothesis and alternative hypotheses using F-test.

2.4. K-Fold Cross Validation

K-fold cross validation is commonly used validation method. In K-fold cross validation method, data is randomly split into K folds and K number of iteration is occurred to train and test the data. K – 1 fold is used for training sets and one set is used test set for each iteration. For each iteration, a new model is trained from training set and model is tested with test set. The resulted errors are averaged to conclude final error for cross validation (Wei, et al., 2019).

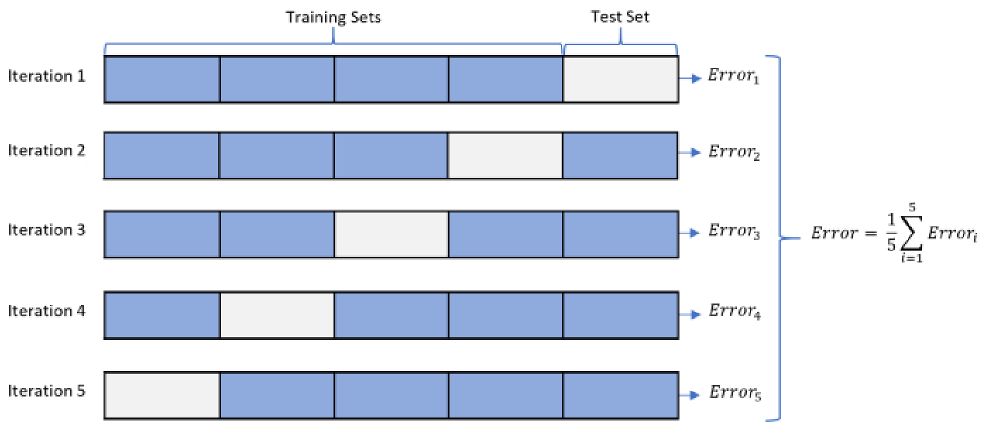

Figure 2.2 5-Fold Cross Validation Representation *(Patro, 2021)*

2.5. Linear Regression

The following equation is defined for simple linear regression model where single predictor variable and response variable is available. β_0 represents the intercept and β_1 represent slope of the model equation where β_0 and β_1 are also defined as unknown regression coefficients (Montgomery & Runger, 2014).

$$Y = \beta_0 + \beta_1 x + \epsilon \qquad (2.6)$$

where Y is the response variable,

x is the predictor variable

ϵ is a random error.

In linear regression fitting problems, the aim is minimizing residual error which is obtained from random errors. $\hat{\beta}_0$ and $\hat{\beta}_1$ are least square estimators of β_0 and β_1. In other words, instead of one of the points in the sample data, regression coefficients for each point are included separately in $\hat{\beta}_0$ and $\hat{\beta}_1$.

For calculation of least square estimates of intercept and slope in linear regression models, following two equations are used.

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\, \bar{x} \tag{2.7}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} {x_i}^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}} \tag{2.8}$$

where $\bar{y}$ and $\bar{x}$ are mean of the element from 1 to n in x and y random variables.

where $\bar{y}$ and $\bar{x}$ are mean of the element from 1 to n in x and y random variables.

Then, estimated regression line can be obtained from following equation (Montgomery & Runger, 2014).

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1\, x \tag{2.9}$$

Each observation satisfies the following equation.

$$y_i = \hat{\beta}_0 + \hat{\beta}_1\, x_i + \epsilon_i \tag{2.10}$$

Where i=1,2, …, n.

$\epsilon_i = y_i - \hat{y}_i$ is also known as residual. The aim is, again, minimizing sum of the residuals to obtain best fitting model. The phenomenon can be observed in the Figure 2.3.

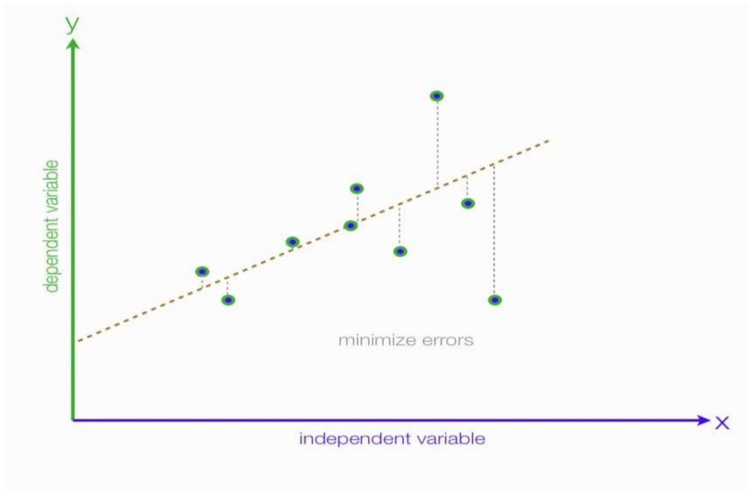

Figure 2.3 Best Linear Regression Model Representation and Residuals

However, most of the problem comprises of more than one predictor variable. These type of regression models are named as multiple linear regression models and generally requires computational power for predictions (Montgomery & Runger, 2014). In general, one response variable can be predicted by using k number of independent variables. The model is defined as multiple linear regression model with k regressor variables (Montgomery & Runger, 2014). Following equation represents the multiple linear regression model. The analogy described in simple linear regression needs to be applied.

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + + \beta_k x_k + \epsilon \qquad (2.10)$$

2.6. Ridge Regression

The idea behind the ridge regression is that firstly, arbitrary two points are chosen, and one line is drawn. According to these two points the line overfit to these two points with zero minimum sum of squared residuals however there is high variance between the first predicted line and other points. Then, by moving away from the points a little, some amount of bias is added to model but, in return, large amount of variance is dropped. The main difference of ridge regression from least squares is that it minimizes squared residuals and ridge parameter lambda multiplied by the square of the slope instead of merely sum of the squared residuals. Then the minimizing element is the following for ridge regression.

$$minimizing\ element = \lambda \times slope^2 \qquad (2.11)$$

In the following equations X is the design matrix. The equation 2.12 gives the least squares estimate. The least square is highly sensitive to random errors where high variances are produced as it is mentioned above (Hoerl & Kennard, 1970).

$$\hat{\beta} = (X^T X)^{-1} X^T y \qquad (2.12)$$

Ridge regression estimates regression coefficient using following formula (Hoerl & Kennard, 1970).

$$\hat{\beta} = (X^T X + kI)^{-1} X^T y \qquad (2.13)$$

where

k is the ridge parameter,

I is the identity matrix.

2.7. Lasso Regression

Least absolute shrinkage and selection operator (LASSO) is very similar regression method to the ridge regression. The main difference between them is taking absolute of slope term.

$$minimizing\ element = \lambda \times |slope|^2 \tag{2.14}$$

Lasso function, used in this project, solves the following problem (Tibshirani, 1996).

$$min_{\beta_0,\beta}\left(\frac{1}{2N}\sum_{i=!}^{N}(y_i - \beta_0 - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right) \tag{2.15}$$

where value of λ given and represents nonnegative regularization parameter,

N is the number of observations,

y_i is the response ith observation,

x_i is ith data, vector of length p,

The parameters β_0 and β are a scalar and a vector of length p, respectively (Tibshirani, 1996).

2.8. Regression Tree

Regression trees are type of decision trees. As the decision trees there are root nodes, internal nodes and leaves. However, in decision trees, the leaves nodes consist of true or false situations or discrete categories. On the other hand, regression trees can consist of numerical situations. Regression trees are good at handling complicated data compared to previous explained models.

The node splitting rules for the algorithm used in this project are the followings. The algorithm firstly calculates mean squared error for the responses (Breiman, et al., 1984).

$$\varepsilon_t = \sum_{j\epsilon T} w_j\left(y_j - \bar{y}_t\right)^2 \tag{2.16}$$

where node t,

w_j is the weight of the observation j,

T is the set of observations.

Then, the algorithm predicts the probability that an observation is in node t by using following formula (Breiman, et al., 1984).

$$P(T) = \sum_{j\epsilon T} w_j \tag{2.17}$$

The algorithm sorts x_i in the ascending order. Sorted predictors split the cut point.

Then, the algorithm decides on splitting say by taking into considering maximizing reduction in MSE for all splitting candidates (Breiman, et al., 1984).

2.9. Neural Network

Neural network or artificial neural network (ANN) is subject of machine learning and highly related with deep learning. Artificial neural network takes its name from the similarity between working mechanism of the neurons in the algorithm and brain (IBM , 2020).

The idea behind the artificial neural networks is very similar to working mechanism of neurons in the brain. In the ANN, there are input layer, output layer and between them, there are one or more hidden layers which a black box how it works after it trained. Each node or neuron pass information to the other neurons in the next layer according to associated weight and threshold. If the threshold is exceeded, information is passed and the connection between two neurons strengthens. If the threshold is not exceeded, any information pass to the next layer (IBM , 2020).

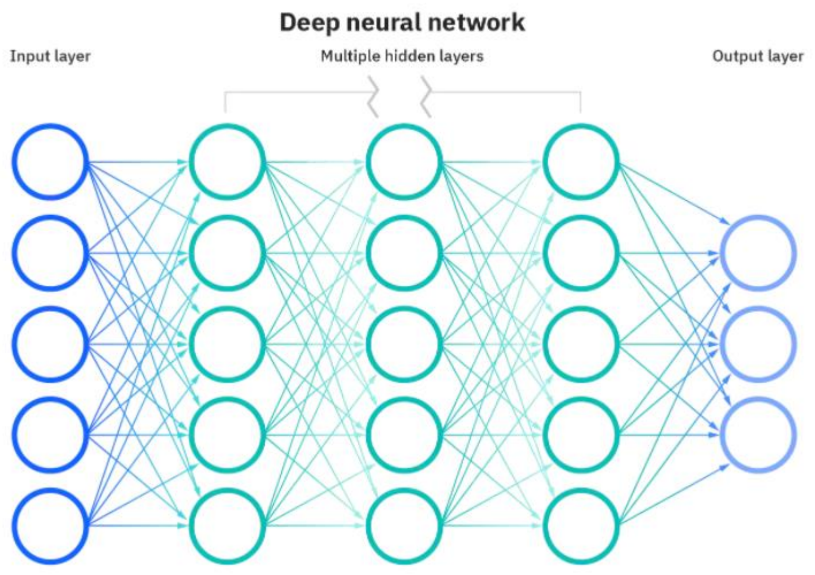

Figure 2.4 Typical Neural Network Representation *(IBM , 2020)*

The neural network can be approximated like that each individual node has its own linear regression model. Then, the formula can be like the following.

$$\sum_{i=1}^{m} w_i x_i + bias \tag{2.18}$$

Then, according to value is lower or higher the threshold, the algorithm pass or not pass the information.

2.10. Errors: R^2 and RMSE

Root mean square error (RMSE) is method to calculate error of model predictions. The following equation is standard formula for RMSE (Moody, 2019).

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.19}$$

where

$\hat{y}_i$ represents the predicted values,

y_i is represent the actual values,

n is the number of observations.

R^2 is the value which indicates how well the model fits the data. It lies between 0 and 1. Closer to 1 means better fit and vice versa. It also known as coefficient of determination. R^2 can be calculated using following formula (Montgomery & Runger, 2014).

$$R^2 = 1 - \frac{SS_E}{SS_T} \tag{2.20}$$

where

SS_E is sum of squares error,

SS_T is sum of squares total.

3. PRE-PROCESSING

In this project, the raw dataset is composed of 22 independent variables and 3 dependent variables with 2189 observations. Only one of the dependent variables is used for modelling the machine learning algorithms which is energy density. The Table 3.1 summarizes code names and real names of features and response variables.

Table 3.1 Features and response variable list

	Code	**Real Name**
Features	ElectMG	Electrode Material Group
	HeatT	Heat Treatment Temperature
	SSA	Specific Surface Area
	IDIG	D- Raman Peak and G- Raman Peak Ratio
	Nconcentration	Nitrogen Concentration
	Oconcentration	Oxygen Concentration
	Pconcentration	Phosphor Concentration
	Bconcentration	Boron Concentration
	Porosity	Porosity
	Pore volume	Pore Volume
	PreparationMG	Preparation Method for Electrode
	ElectrolyteType	Electrolyte Type
	Electrolyteconcentration	Electrolyte Concentration
	Normalized ScanRate	Normalized Scan Rate
	AbsolutePotentialWindow	Absolute Potential Window
	Separator	Separator
	CurrentCollector	Current collector
	SaltAnionVolume	Salt Anion Volume
	SaltCationVolume	Salt Cation Volume
	SolventDipoleMoment	Solvent Dipole Moment
	SolventVolume	Solvent Volume
	Binder conc	Binder Concentration
Response Variable	Energy	Energy Density

3.1. SEM Image Data Collection

Image processing on SEM images is done to enhance the quality of dataset. Pore volume and porosity are two feature columns of the dataset. Although 1561 observation was directly collected from articles, 628 of the observations were missing when the dataset first prepared for porosity and pore volume columns. The missing cells are filled by averaging available cells. However, this approximation is open to development. Therefore, the new technique where the SEM images are analyzed to approximate porosity of the samples. The open-source MATLAB code is used to calculate porosity of the substances. As a result of the analysis, 84 of the 628 predicted porosity and pore volume observations are replaced with the results that come from SEM image processing approximations.

3.1.1. Algorithm and Results

The algorithm that is used in the project predicts the porosity and pore volume values of the samples according to SEM images. However, pore size distribution information is not used in this project. The code applies some image processing techniques. The algorithm and results are explained in this section.

The algorithm results in 3 different images which are depth map, pore space segmentation and binary segmentation represented in Figure 3.1. The algorithm starts with reading image as an input data. The image is perceived as a matrix where every number of the matrix represents the color and tone of the pixels. Then algorithm continues with transforming image completely gray scale. Next, the algorithm decides on single threshold value to convert two level image efficiently. In the matrix of the images 0 represents black and 255 represents white color. Therefore, in the algorithm, above the threshold value treated as white and below the threshold treated as black. This approximation also known as binarization. In the Figure 3.1.d binary segmentation also can be seen. After binarization step, image matrix contains only zeros and ones where zeros represent black and ones represent white. Then, the porosity calculation is simply ratio of number of black pixels to the whole pixels because black pixels are more deep parts of the image from the 3D perspective. To do that, mean of the binarized matrix is subtracted from one since the porosity is the portion of the void space but mean of the binarized matrix is portion of the solid space.

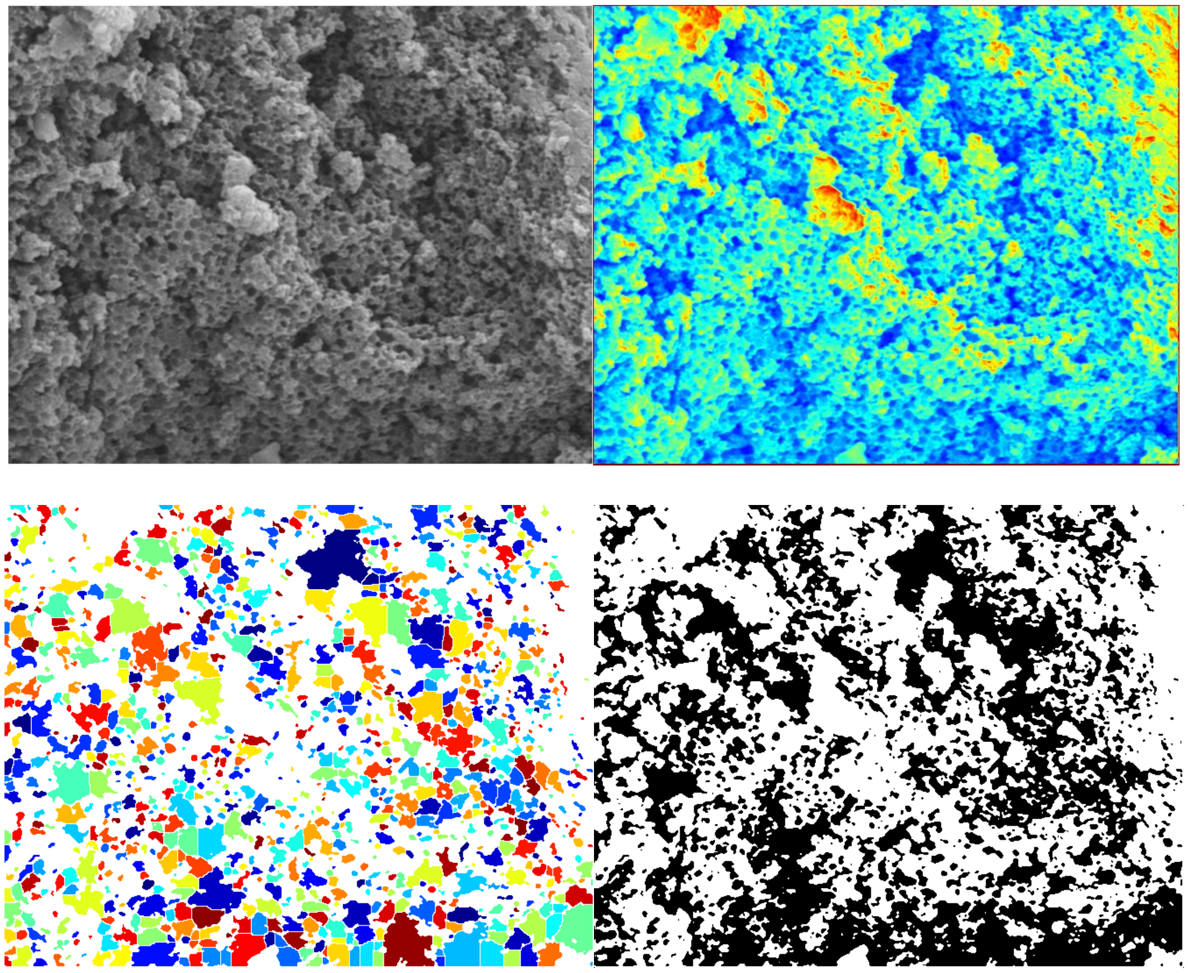

Figure 3.1 SEM image processing results a) Original image b) Depth map visualization c) Pore space segmentation d) Binary segmentation

Beside the binarization, depth map and pore space segmentation are other results that is obtained from the analysis. As it can be seen from the Figure 3.1.b more deeper parts of the image is represented with blue color, on the other hand, more shallow parts are showed with yellow and red colors. In the Figure 3.1.c, pore space segmentation is can be seen. This demonstration is more likely used for average pore radius. However, since the error between pore radius that is obtained from MATLAB algorithm and taken from articles is enormously high, the average pore radius calculations are not taken into consideration.

The last part for more clarification is optimization of the results. The algorithm gives flexibility to determine optimum number of intensity levels in image by changing number called, N. If the user thinks porosity is overestimated, N can be increased to prevent overestimation. On the other hand, underestimation can be handled by simply decreasing N.

3.2. Filtering Data

In this project, data is, firstly, filtered according to outliers in the feature called porosity. Then, data is filtered according to accuracy of model predictions which come from testing the models and RMSE values come from linear regression models. Data is shrunk iteratively until the RMSE results do not change too much because decreasing amount of data without contributing accuracy of the model is unfavorable for training model. The larger data provides better trained models. The result of the unfiltered model can be observed from the Figure 3.2. It is obvious that most of the observations are accumulated in the section lower than 100. The figure contains 2159 observations and test RMSE of 36.42.

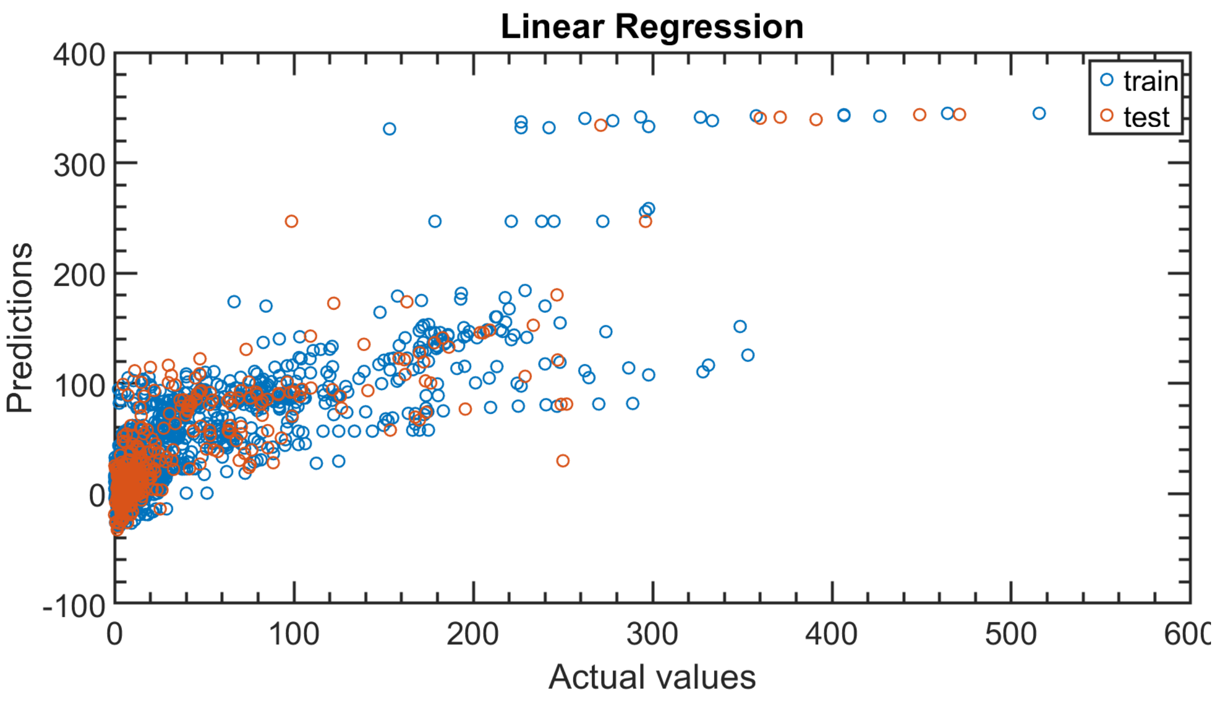

Figure 3.2 Actual values versus predicted values without any filtering data

The problem, however, is not solved by bounding data according to value of the data. Instead, the while loop is written to shrink data where percentage number of observations is changed. This is better approximation because it decreases bias of the human interpretation. Then, the data is shrunk until the 70% the data is hold and 30% of the data is removed according to response variable namely, energy density. Until this point, decreasing data does not change RMSE too much but it results in losing data.

Table 3.2 Filtering data according to RMSE values

Percentage of the Data Maintained	Linear Regression Test RMSE	Regression Tree Test RMSE	Threshold Value
100	36.42	20.82	Not Exist
95	25.23	14.13	178
90	14.81	10.77	107
85	11.42	8.31	80
80	7.92	7.39	58
75	5.83	4.32	42
70	5.37	3.96	31

In Table 3.2, manually conducted data filtering according to response variable is summarized. It is obvious that there is significant decrease in RMSE when 30 % of data is eliminated. The last column represents the threshold value where rows with lower than that point of energy density is eliminated. It is important to observe that there is not large change in threshold column when moving from 75 % to 70 %. This means that most of the energy density data is accumulated lower than 31. The RMSE values and percentage of the maintained data graph is given in the Figure 3.3 which is obtained from Table 3.2.

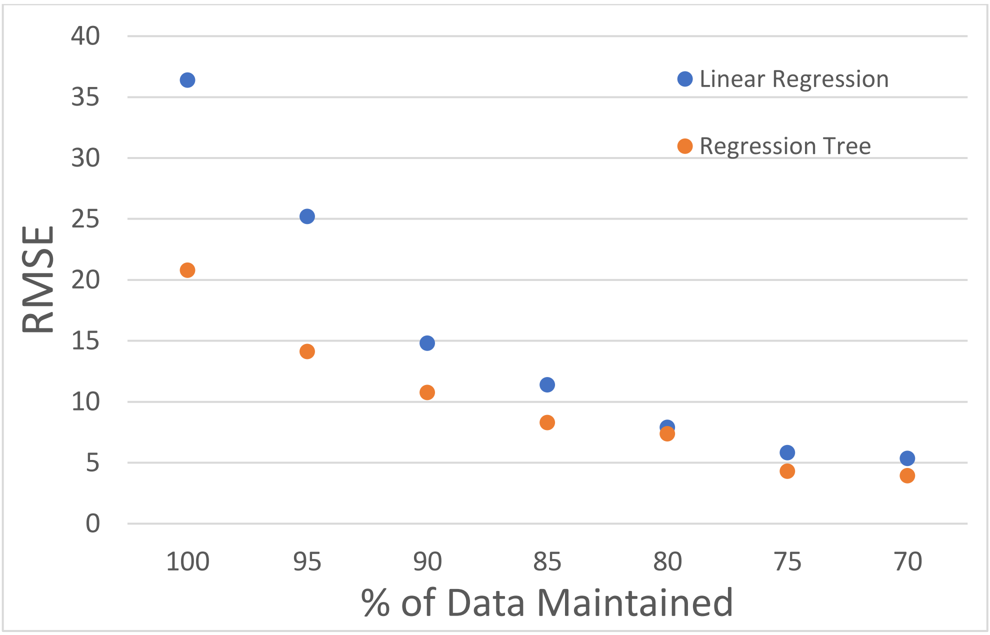

Figure 3.3 RMSE result according to percentage of data maintained

Lastly, in the Figure 3.4, 30 % of the data is eliminated according to response variable. As it can be seen from the Figure values are well distributed according to linear regression model.

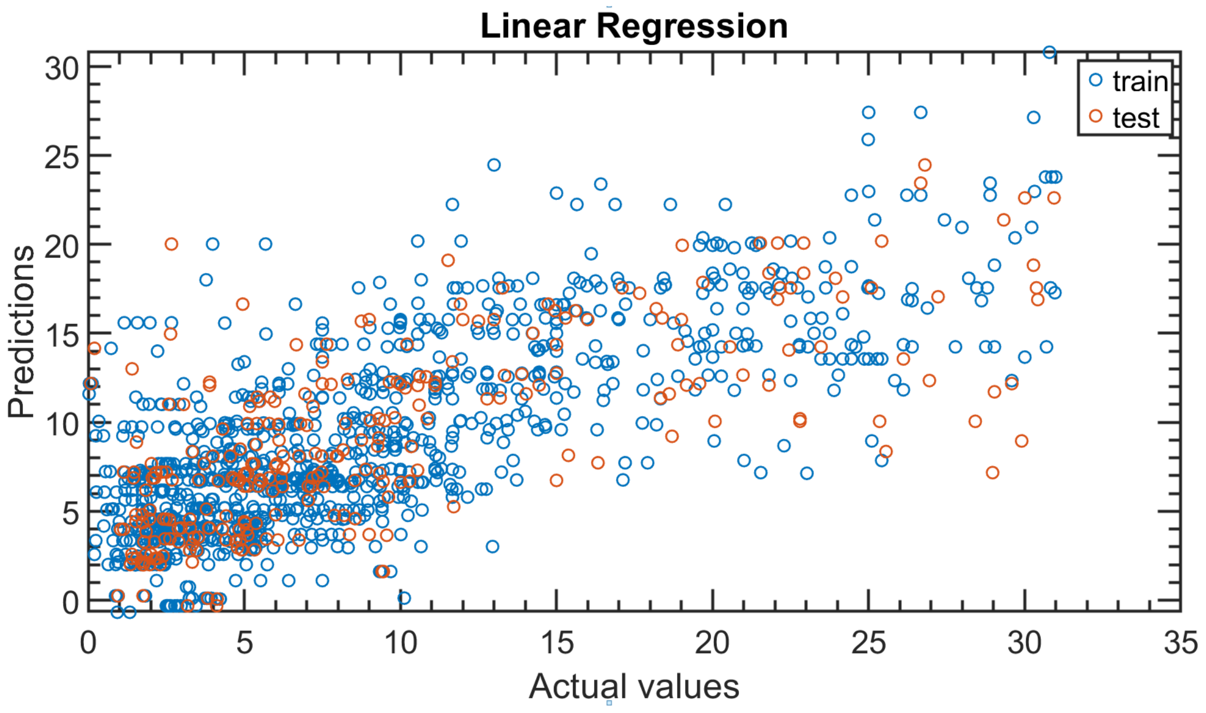

Figure 3.4 The predicted values versus actual values graph for 70 % of the data maintained

3.3. Normalization & Dummy Coding

In this project, numerical data is normalized, and categorical data is applied to the dummy coding. In normalization, numerical values are bounded between 0 and 1. This method prevents failures that are resulted from different units in one column. Also, model is more accurately trained with normalized data. It balances the effect of larger and smaller data.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Dummy coding, on the other hand, a way to handle with categorical data. Most of the regression models do not approve categorical data for regression. Therefore, they need to be converted numerical values. For this purpose, dummy coding is applied to the categorical columns. For example, if one categorical column consists of 4 different categories, in dummy coding, it is converted to the 4 different columns where each of the column is full of 1 for the related categorical value and other cells are filled with zero. Following table is an example for the clear understanding.

Table 3.3 Representative configuration for dummy coding

Categorical Feature	Column For A	Column For B	Column For C	Column For D
A	1	0	0	0
A	1	0	0	0
D	0	0	0	1
B	0	1	0	0
C	0	0	1	0
A	1	0	0	0
D	0	0	0	1

4. FEATURE SELECTION

Feature selection is important step for further analysis. In this section, features will be eliminated according to results of the feature selection methods. Three methods are applied in this section: correlation matrix, F-test and trial & error method.

4.1. Correlation Matrix and Heat Map

Correlation matrix is formed by using formulas defined in the Section 2.2. Firstly, correlation between two features is calculated. Then, all of the calculated correlation coefficients inserted into correlation matrix. As it can be seen from Figure 4.1, some of the feature pairs are negatively correlated. The diagonal, which is formed with coefficient one, divides correlation matrix to two parts. These two parts are replica of each other and do not provide different information. The other important thing is that heat map is colored with the tones of blue. As it can be seen from the figure darker tones indicates higher positive correlation, on the other hand, lighter tones represent less correlation or negative correlation. In the following figure, only numerical predictors are evaluated for correlation matrix. In other words, categorical variables is not included.

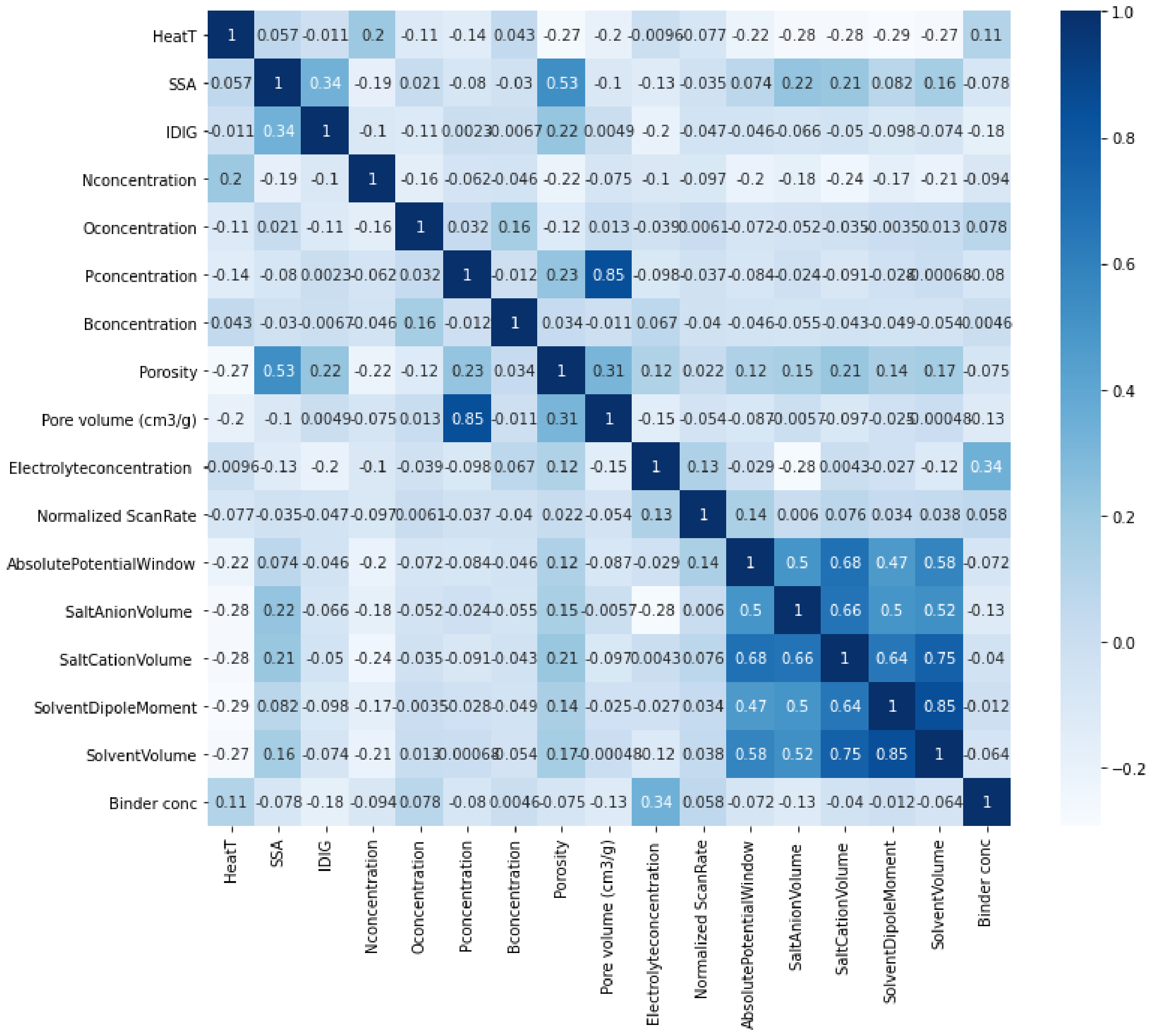

Figure 4.1 Correlation matrix results and heat map of the features

From the Figure 4.1, one can conclude that there is strong correlation between following pairs:

- Pore volume and P concentration with correlation of 0.85
- Solvent cation volume and absolute potential window with correlation of 0.68
- Solvent cation volume and salt anion volume with correlation of 0.66
- Solvent cation volume and solvent dipole moment with correlation of 0.64
- Solvent cation volume and solvent volume with correlation of 0.75
- Solvent volume and solvent cation volume with correlation of 0.85

After the analysis, it is concluded that correlation coefficients are not sufficiently large to eliminate any feature.

4.2. Filter Type Feature Selection: F-Test

In this section, one of the filter type feature selection methods is preferred namely, F-Test. This method ranks the features according to predictor importance results which are resulted in F-Test results. The function is available in MATLAB which takes all of the features and response variable as inputs and returns the predictor importance results of features by relating features with response variables using F-test.

In the Figure 4.2, predictor importance results according to F-test are reported. As it can be seen from the figure, features can be grouped into 4 different categories if it is generalized. The first category is the feature with higher than 200 points namely, separator. The second category consists of predictors that are between 200 and 125 scores. The elements of the second category are solvent volume, electrolyte type, solvent dipole moment, salt cation volume and potential window. The third category lies between 125 and 40 scores with the elements of preparation MG, porosity, pore volume, ID/IG ratio, current collector. The fourth category consists of features between 40 and 15 scores which are N concentration, electrolyte concentration, electrode MG, O concentration, binder concentration, SSA, and Heat T. The fifth category consists of features with less than 15 scores which are P concentration, normalized scan rate and B concentration. The elimination is conducted for the fourth category. In other words, by using F-test feature selection method, P concentration, normalized scan rate and B concentration are eliminated for the further steps of the analysis.

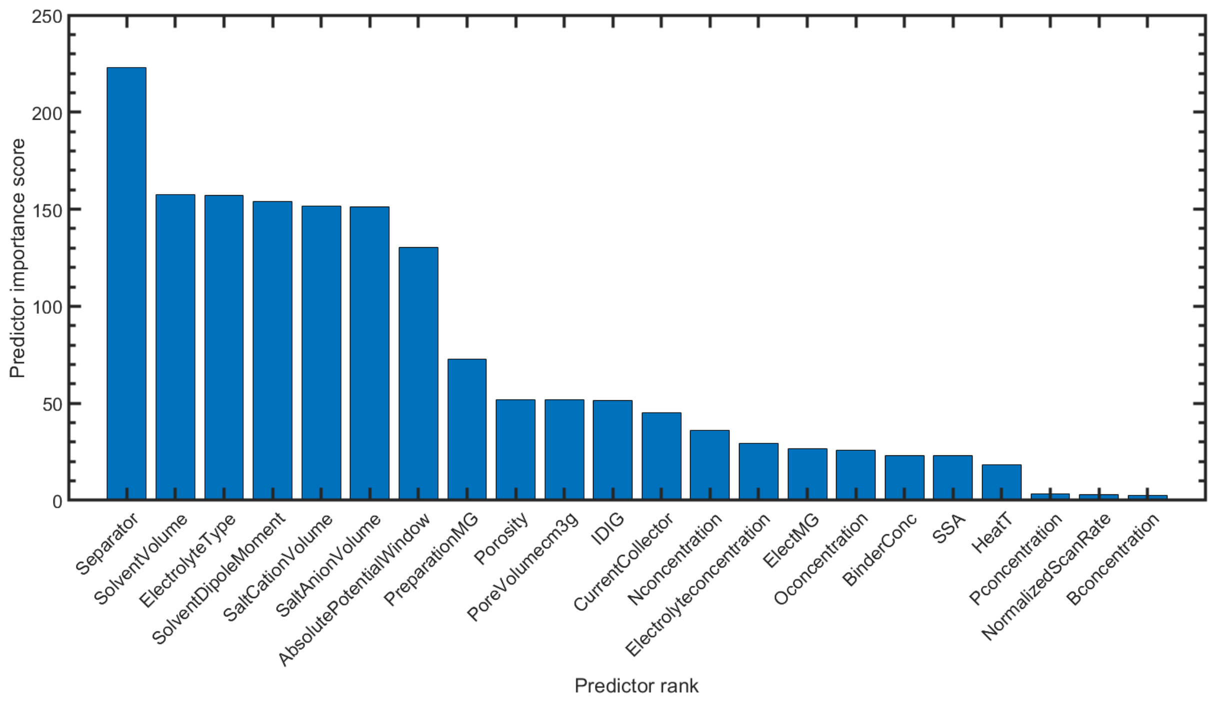

Figure 4.2 Rank of the features according to predictor importance results

4.3. Trial and Error Method

The last feature selection method is named as trial-and-error method. In the Table 4.1, first row, named as all features, consists of RMSE values for 5 different models where all the features remained after F-Test feature selection method is applied. Then, in the each following row, one of the features is removed and models are trained with the training data, again. The minus in front of the features indicates the feature is removed. The RMSE are calculated by testing models with unseen testing data and each of the error is reported in the table. The aim is that observing change in the error when one of the features is eliminated. If the error increases when the feature is removed from the training and testing data, it means that that feature is important to train successful model. On the other hand, if the error decreases when the feature is eliminated, one can conclude that the feature does not much important for training the model *(Nascimento, et al., 2000).*

Table 4.1 RMSE Results According to Elimination of Features

	Root Mean Squared Error				
Features	**Linear Regression**	**Lasso Regression**	**Ridge Regression**	**Regression Tree**	**Neural Network**
All Features	5.0253	4.8308	5.0233	3.9170	3.6983
-Heat T	5.0214	4.8351	5.0219	3.6739	4.2678
-SSA	5.1657	5.0062	5.1663	4.0361	4.1242
-IDIG	5.0802	4.9325	5.0801	3.9275	4.1200
-N Concentration	5.0643	4.9115	5.0647	3.7589	4.4974
-O Concentration	5.0183	4.8312	5.0281	3.6636	4.1665
-Porosity	5.0032	4.8229	5.0074	3.7748	3.9573
-Pore Volume	4.9974	4.8405	5.0197	3.6411	4.0805
-Electrolyte Concentration	5.0988	4.8962	5.0978	3.7907	4.2861
-Absolute Potential Window	5.2498	5.0273	5.2524	3.9560	4.0126
-Salt Anion Volume	5.0004	4.8413	5.0020	3.8218	4.1896
-Salt Cation Volume	5.0190	4.8963	5.0199	3.7958	4.2326
-Solvent Dipole Moment	5.0674	4.8789	5.0698	3.7208	3.8870
-Solvent Volume	5.1866	4.8907	5.0739	4.0393	4.1894
-Binder Concentration	5.0323	4.8538	5.0318	3.6654	3.7445
-Elect MG	5.0637	4.8854	5.0626	3.7469	4.5221
-Preparation MG	5.2444	5.1532	5.2442	3.7080	4.2128
-Electrolyte Type	5.0315	4.8425	5.0306	3.8413	4.1675
-Separator	5.2076	5.0415	5.2102	3.8505	5.3960
-Current Collector	5.0444	4.9015	5.0443	3.7158	6.1616

However, since the errors are very close to each other, it is hard to observe change in the RMSE values. Therefore, following table is formed to see error changes easier. The math behind creating values in Table 4.2 is showed in the following formula.

$$\Delta RMSE = \frac{RMSE_i - RMSE_{all}}{RMSE_{all}} \times 100 \qquad (4.1)$$

where

$\Delta RMSE$ is the percentage differences in RMSE,

$RMSE_i$ is the RMSE value when i$_{th}$ feature is eliminated,

$RMSE_{all}$ is the RMSE value when all features are included.

Table 4.2 Percentage Differences in RMSE for Each Eliminated Feature

	Percentage Differences between RMSE values				
Features	**Linear Regression**	**Lasso Regression**	**Ridge Regression**	**Regression Tree**	**Neural Network**
Heat T	-0.08	0.09	-0.03	-6.21	15.40
SSA	2.79	3.63	2.85	3.04	11.52
IDIG	1.09	2.11	1.13	0.27	11.40
N Concentration	0.78	1.67	0.82	-4.04	21.61
O Concentration	-0.14	0.01	0.10	-6.47	12.66
Porosity	-0.44	-0.16	-0.32	-3.63	7.00
Pore Volume	-0.56	0.20	-0.07	-7.04	10.33
Electrolyte Concentration	1.46	1.35	1.48	-3.22	15.89
Absolute Potential Window	4.47	4.07	4.56	1.00	8.50
Salt Anion Volume	-0.50	0.22	-0.42	-2.43	13.28
Salt Cation Volume	-0.13	1.36	-0.07	-3.09	14.45
Solvent Dipole Moment	0.84	1.00	0.93	-5.01	5.10
Solvent Volume	3.21	1.24	1.01	3.12	13.28
Binder Concentration	0.14	0.48	0.17	-6.42	1.25
Elect MG	0.76	1.13	0.78	-4.34	22.27
Preparation MG	4.36	6.67	4.40	-5.34	13.91
Electrolyte Type	0.12	0.24	0.15	-1.93	12.69
Separator	3.63	4.36	3.72	-1.70	45.90
Current Collector	0.38	1.46	0.42	-5.14	66.60

Absolute value of the equation is not calculated in purpose because it means that feature is important for the model when the value is positive, vice versa. Then, the features with first seven largest values are kept and rest of them removed for the further calculations. The kept features are represented with yellow color in the table.

5. PARAMETER TUNING & MODEL TRAINING

The model training in this project is conducted twice for the same data. Firstly, the data is divided to 2 parts with the ratio of 90 % and 10 %. The 10 % of the data is used for validation data which is used for testing in the parameter tuning. The part with 90 % is divided into training and test data according to 5-fold cross validation method. One alternative for splitting data once from arbitrary point which also known as hold-out method. However, if the data would be divided according to this method, results would heavily depend on from where the data is split. On the other hand, 5-fold cross validation method eliminates this effect by increasing randomness.

80 % of 90 % of the data is split for training data which is equals to 72 % of the total data with 5-fold cross validation method. The remaining 18 % is kept for the testing data. The data is the 72 % for both training model in parameter tuning and the latter model training and testing. In summary, data is divided to the three as in the following table.

Table 5.1 Data splitting names and percentages

Data Name	Percentage of the Data	Used Area	Cross Validation
Full Data	100	-	-
Training Data	72	Used for training models	Yes
Testing Data	18	Testing for the final model	Yes
Validation Data	10	Testing the created model for parameter tuning	No

The algorithm is worked as that firstly parameter tuning required models are trained with the training data. The parameter tuning required models are ridge and lasso regression. The for loop is constructed to train data with the training data iteratively and in each iteration, coefficient of lasso or ridge regression is changed. Then, the model prepared for the tuning is tested with validation data. The result where the less RMSE is occurred stored, and the associated coefficient of this iteration is sent to the next model training step. For the regression tree and linear regression parameter tuning is not conducted but for the ANN, it is conducted manually.

Table 5.2 Tuned models and parameters of the models

Models	Parameter Tuning	Tuned Variable	Tuned Values
Linear Regression	No	-	-
Ridge Regression	Yes	λ	4.9
Lasso Regression	Yes	α	1
Regression Tree	No	-	-
ANN	Yes	Hidden Layer Size	25

In the next step, 5 different folds with 5 different training sets and 5 different test sets are used to train and test the models. In other words, 5 different times models are trained with tuned variables, and they are tested with associated testing sets 5 times. Then, the 5 different resulted RMSE is averaged for one final RMSE. An important point is that models are always trained with training set and tested with testing set. Also, it is tested with validation set above. Therefore, prepared models are never seen the testing and validation sets. R^2 error values are also presented.

6. RESULTS AND DISCUSSION

6.1. Linear Regression

This section includes results of the trained linear regression model. In Table 6.1, there are error values of linear regression model. The RMSE value is relatively low which indicates the success of the model. However, R^2 value indicates the accuracy of the model is low because it is away from 1. Low R^2 also indicates high bias for the model. However, these results are logical since linear regression is one of the simplest models it can trained. On the other hand, in Figure 6.1, there is a seen linearity between predictions and actual values. There is only one outlier which is above 30 on the prediction axis. This point affects the outlook of the figure a lot.

Table 6.1 Error values for linear regression

Model Type	Test RMSE	R^2	Train RMSE
Linear Regression	5.2343	0.2838	5.1074

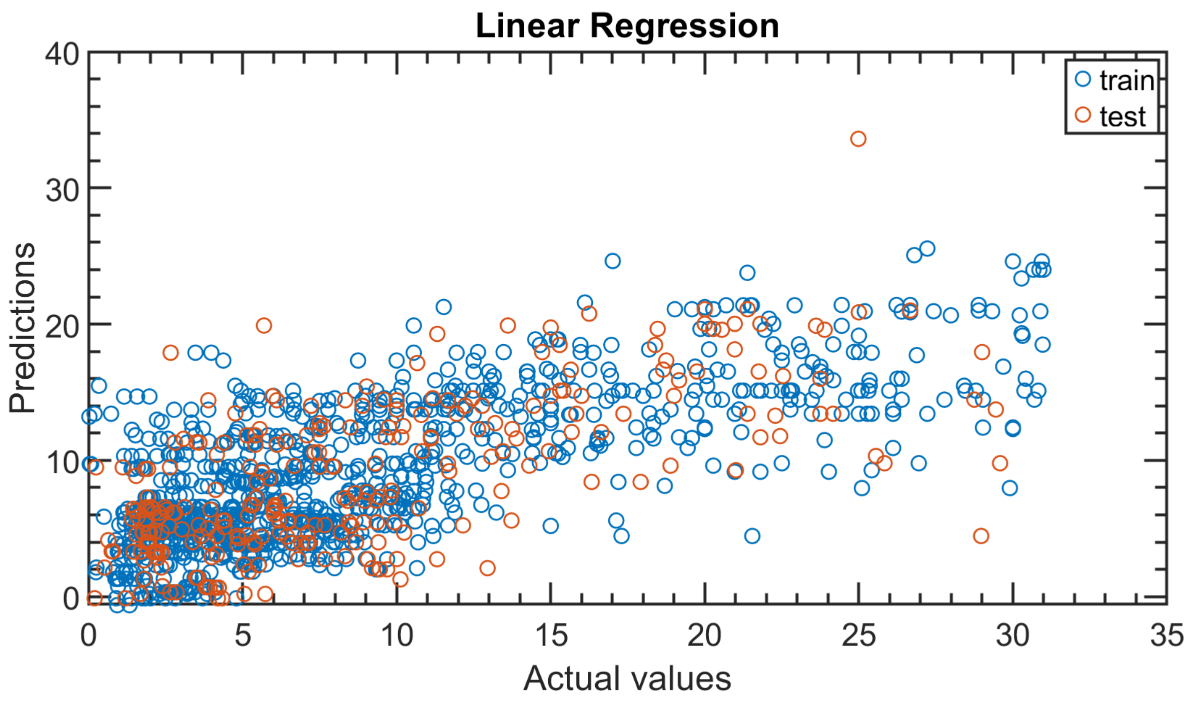

Figure 6.1 Prediction values versus actual values for linear regression

6.2. Ridge Regression

In Table 6.2, the results of ridge regression are very similar to linear regression since both are close regression models. However, in ridge regression RMSE and R^2 results, there is very slight improvement. The Figure 6.2 has very similar distribution to linear regression figure.

Table 6.2 Error values for linear and ridge regression

Model Type	Test RMSE	R^2	Train RMSE
Linear Regression	5.2343	0.2838	5.1074
Ridge Regression	5.2335	0.2839	5.1075

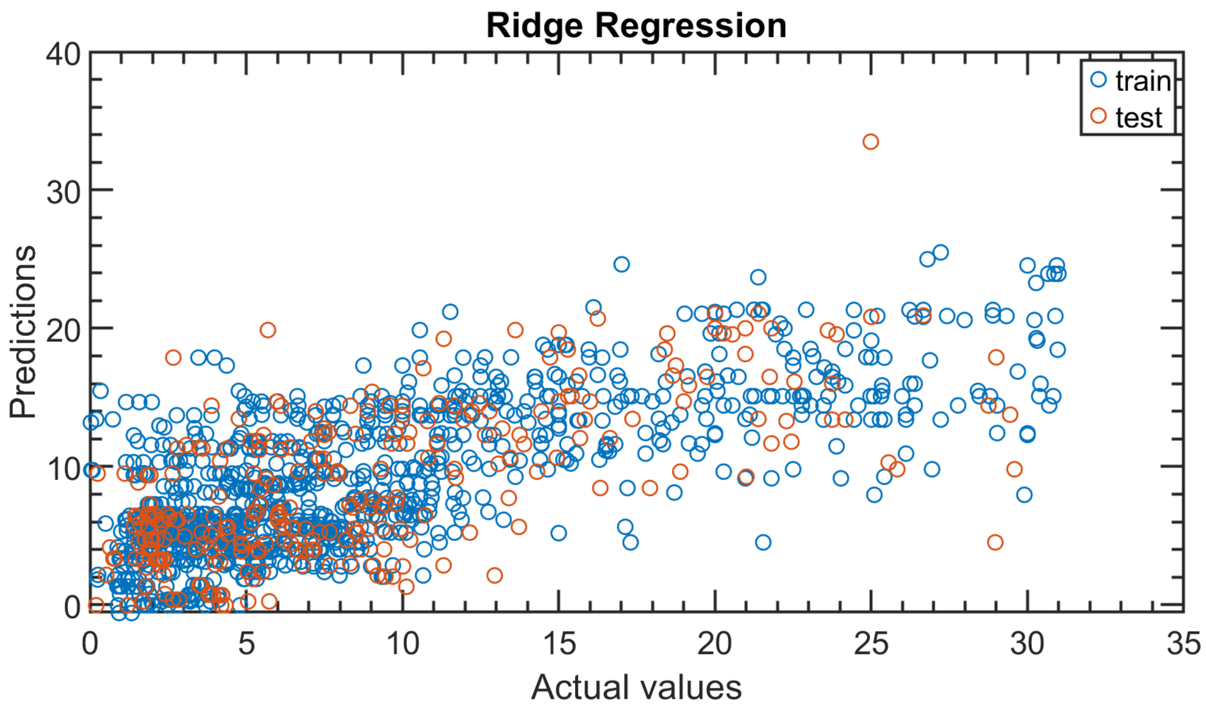

Figure 6.2 Prediction values versus actual values for ridge regression

6.3. Lasso Regression

The error values of lasso regression are relatively better than linear and ridge regression error results. For example, test RMSE of lasso regression is lower than both linear and ridge regression. Similarly, R^2 values are higher for the lasso regression. In Figure 6.2, it seen that there is more dispersed distribution. However, linearity can be still observed from the figure.

Table 6.3 Error values for linear, ridge and lasso regression

Model Type	**Test RMSE**	**R^2**	**Train RMSE**
Linear Regression	5.2343	0.2838	5.1074
Ridge Regression	5.2335	0.2839	5.1075
Lasso Regression	5.1288	0.2989	5.1284

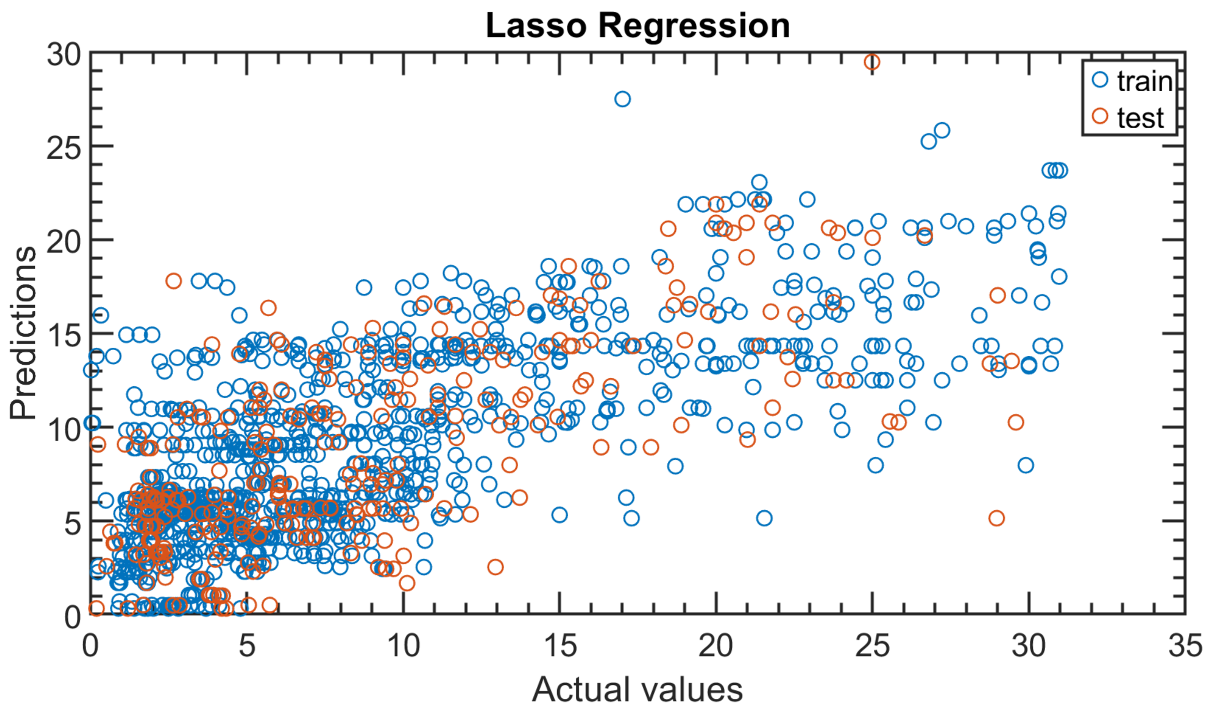

Figure 6.3 Prediction values versus actual values for lasso regression

6.4. Regression Tree

Regression tree results have significant amount of decrease in RMSE compared to previous regression methods. R^2 also approaches to 1 dramatically. In Figure 6.4, there is clear linearity between actual points and predictions. This means that predictions of the model are accurate with less bias. It is sensible that results of the regression tree are better than the previous regression models because regression tree is too far complicated model compared to linear, ridge and lasso regression. However, there would be overfitting since the errors are very small amount.

Table 6.4 Error values for linear, ridge and lasso regression and regression tree

Model Type	Test RMSE	R^2	Train RMSE
Linear Regression	5.2343	0.2838	5.1074
Ridge Regression	5.2335	0.2839	5.1075
Lasso Regression	5.1288	0.2989	5.1284
Regression Tree	3.7315	0.9567	2.9517

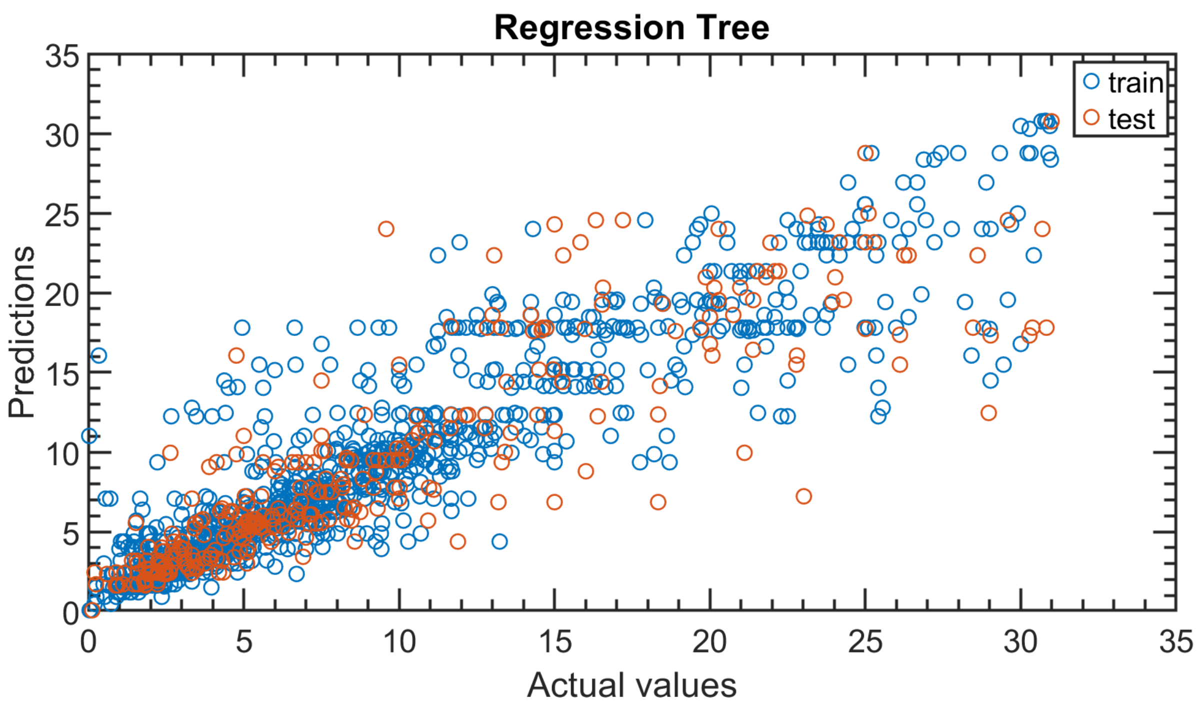

Figure 6.4 Prediction values versus actual values for linear regression

6.5. Artificial Neural Network

Artificial neural network test RMSE is lower than linear, ridge and lasso regression methods. However, the model has higher RMSE than regression tree. This means that regression tree fits the data better than ANN. Another approximation would be that regression tree is overfitting, on the other hand, ANN gives accurate model.

Table 6.5 Error values for linear, ridge and lasso regression, regression tree and ANN

Model Type	Test RMSE	R^2	Train RMSE
Linear Regression	5.2343	0.2838	5.1074
Ridge Regression	5.2335	0.2839	5.1075
Lasso Regression	5.1288	0.2989	5.1284
Regression Tree	3.7315	0.9567	2.9517
ANN	5.010	0.7438	-

In Figure 6.5, created ANN model is represented. There are 25 hidden layers for the prediction of outputs.

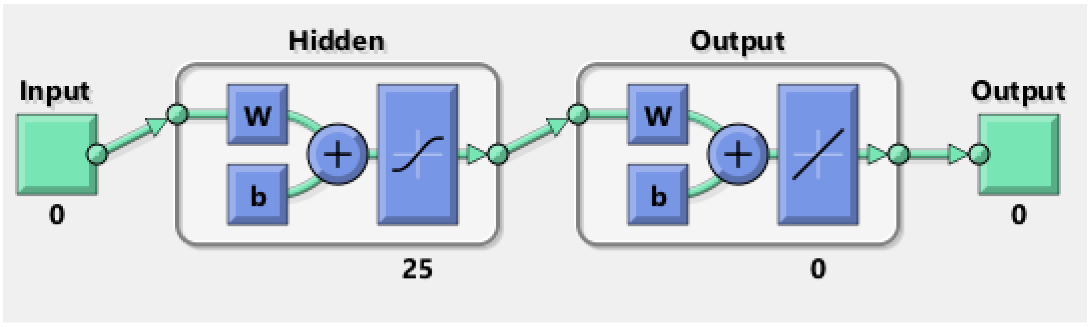

Figure 6.5 Hidden layers representation for ANN

The Figure 6.6 shows that iterations are stopped at 18th iteration where epoch means iteration. The best validation performance occurred at 12th iteration with the value of 16.1432 MSE. The root of the MSE is 4.0179. As it is expected, test error is above from validation and train errors, as well as validation set is above from train data. Since errors are relatively low, one can be concluded that model is appropriate for this dataset.

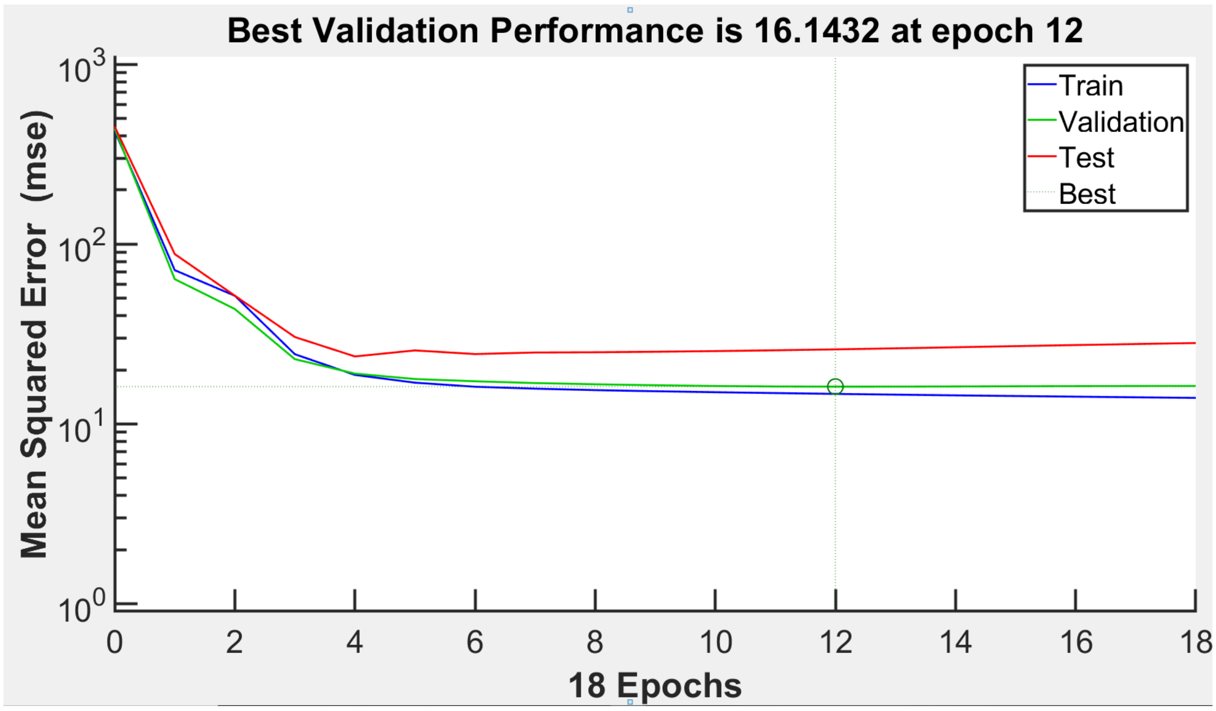

Figure 6.6 Mean squared error changes according to iterations

The Figure 6.8 shows error histogram of the model. Each bin show different error values which are calculated by taking difference of actual value and the predicted value. The x-axis shows number of instances that are available at associated error value. The closeness of the errors to zero indicates that predicted values are close to actual values. In other words, since most of the data is accumulated around zero, the model makes good predictions.

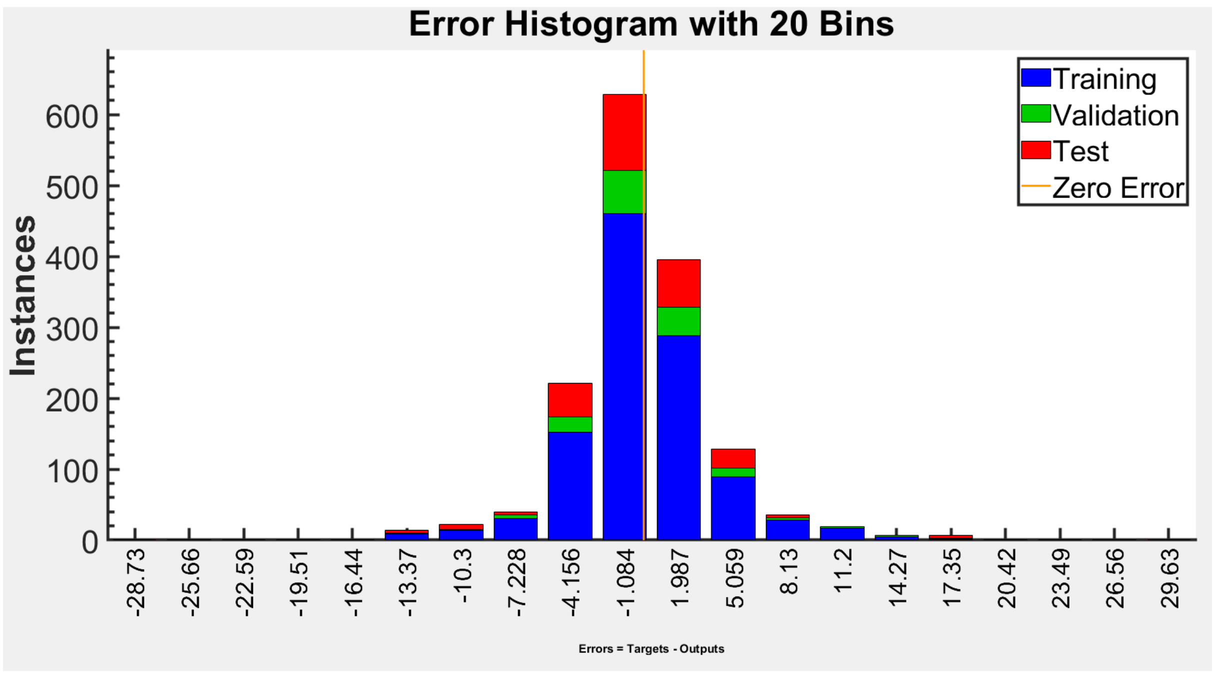

Figure 6.7 Error histogram of training, validation, and test data

In Figure 6.9, the predicted value for the ANN is drawn. The red line show the fit line and the equation of the fit line given in the left side of the figure. R value is given at the top of the figure. The looks good for predicting the testing values. However, some negative predictions are available in the figure. This is not possible for energy density predictions.

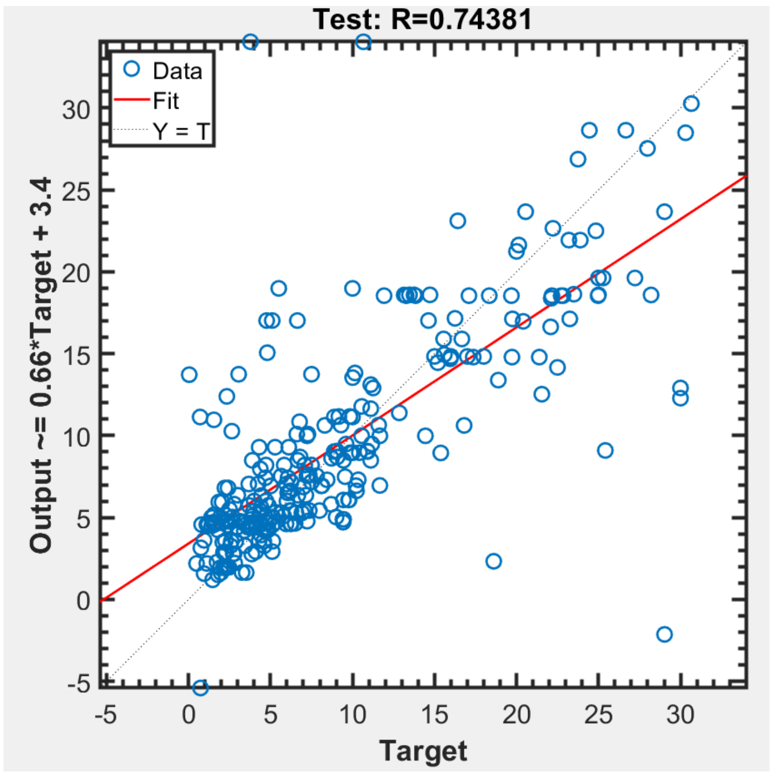

Figure 6.8 Actual values versus predicted values for ANN from test data

The following three figures are representation of the predictions made for validation, training and total datasets, relatively.

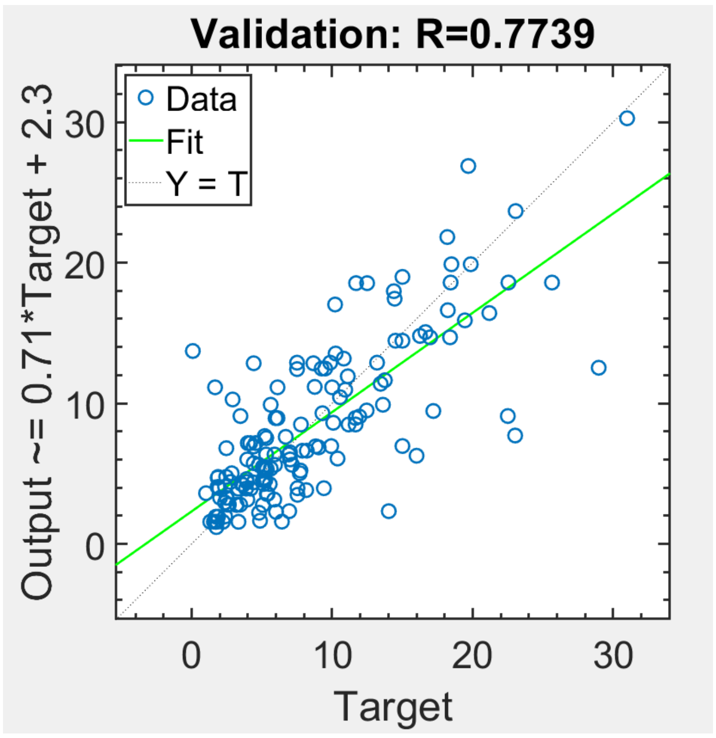

Figure 6.9 Actual values versus predicted values for ANN from validation data

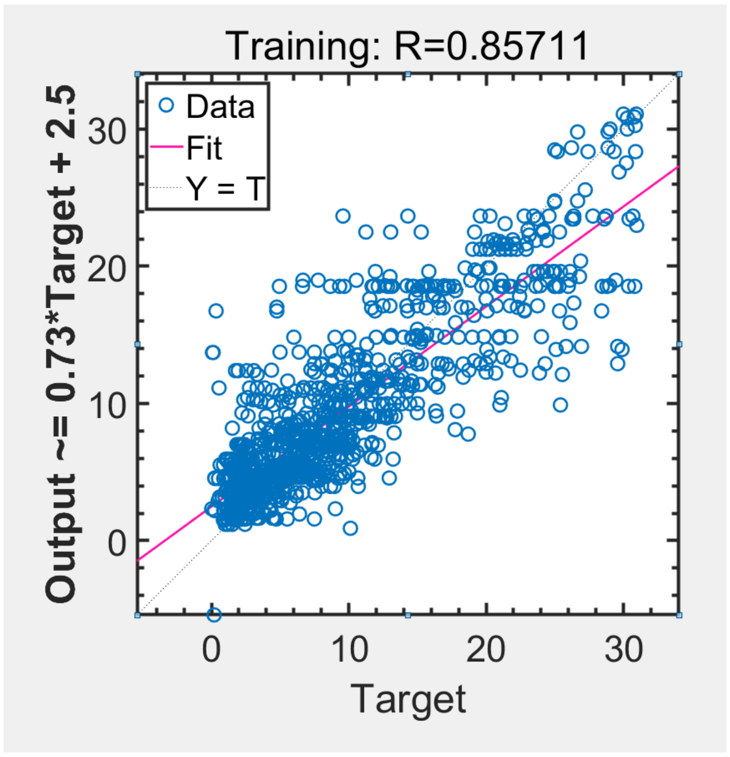

Figure 6.10 Actual values versus predicted values for ANN from training data

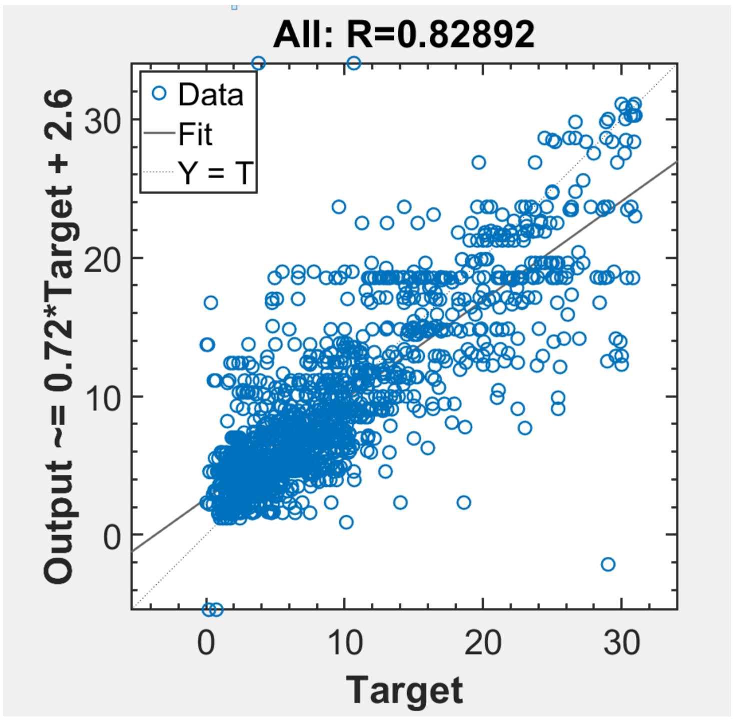

Figure 6.11 Actual values versus predicted values for ANN from all data

7. CONCLUSION

In this project, a machine learning model is trained by using data collected from literature to predict energy density of supercapacitors. It is aimed to save time and money for the material selection processes to design supercapacitor by using machine learning model. One can give inputs to the machine learning model and can expect reasonable performance parameters without doing any real-life experiments. For this purpose, firstly, a raw dataset is taken with 22 independent variables, 1 dependent variable and 2189 observations. Then, 4 pre-processing is applied to the dataset: SEM image processing, filtering data, normalization and dummy coding. With these pre-processing steps, data is enriched and made suitable for machine learning applications. Then, features are eliminated using techniques that are correlation matrix, F-Test and trial error method. After these two steps, data is shrunk to the 7 features and 1521 observations. Then, for modelling and parameter tuning, 5-Fold cross validation is applied to the data. Five different models are trained and tested. These are linear regression, ridge regression, lasso regression, regression tree and artificial neural network. The RMSE results of the models are 5.2343, 5.2335, 5.1288, 3.7315 and 5.010, relatively. The R^2 results of the models are 0.2838, 0.2839, 0.2989, 0.9567 and 0.7438. In summary, project achieves its aim which is training successful models for supercapacitor design.

8. FUTURE THOUGHTS

In this report only one of the performance parameters of supercapacitors is predicted which is energy density. However, there are also two different response variable that can be used to train new models. These response variables are power density and capacitance.

Another contribution can be done to this project is that checking the variability of the models with different datasets. In other words, by collecting new data, model can be tested for different datasets to ensure model has low variability for these new datasets.

9. REFERENCES

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R., 1984. *Classification and Regression Trees.* s.l.:CRC Press.

Conway, B. E., 1999. *Electrochemical Supercapacitors: Scientific Fundamentals and Technological Applications.* New York: Springer Science+Business Media.

Deringer, V. . L., 2020. Modelling and understanding battery materials with machine-learning-driven atomistic simulations. *JPhys Energy,* Volume 2.

Gu, W. & Yushin, G., 2013. Review of nanostructured carbon materials for electrochemical capacitor applications: advantages and limitations of activated carbon, carbide-derived carbon, zeolite-templated carbon, carbon aerogels, carbon nanotubes, onion-like carbon, and graphene. *WIREs Energy and Environment.*

Hoerl, A. E. & Kennard, R. W., 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics,* Volume 12, pp. 55-67.

IBM , C. E., 2020. *Neural Networks.* [Online] Available at: https://www.ibm.com/cloud/learn/neural-networks [Accessed 06 07 2021].

Miller, J. R. & Simon, P., 2008. Electrochemical capacitors for energy management. *Science,* Volume 321, p. 651–652.

Montgomery, D. C. & Runger, G. C., 2014. *Applied Statistics and Probability for Engineers.* 6th ed. s.l.:John Wiley & Sons Singapore Pte. Ltd.

Moody, J., 2019. *towards data science.* [Online] Available at: https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e [Accessed 5 july 2021].

Naik, K., 2020. *Complete-Feature-Selection.* [Online] Available at: https://github.com/krishnaik06/Complete-Feature-Selection [Accessed 2021].

Nascimento, C. A. O., Giudici, R. & Guardani, R., 2000. Neural network based approach for optimization of industrial chemical processes. *Computers & Chemical Engineering,* Volume 24, pp. 2303-2314.

Patro, R., 2021. *towards data science.* [Online] Available at: https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b
[Accessed 04 July 2021].

Rabbani , A. & Salehi, S., 2017. Dynamic modeling of the formation damage and mud cake deposition using filtration theories coupled with SEM image processing. *Journal of Natural Gas Science and Engineering,* Volume 42, pp. 157-168.

Salanne, M., Rotenberg, B., Naoi, K. & et al., 2016. *Efficient storage mechanisms for building better supercapacitors.* s.l.:Nat Energy.

Serway, R. A. & Jewett, J. W., 2013. *Physics for Scientists and Engineers with Modern Phsics.* 9th ed. s.l.:Cengage Learning.

Simon , P. & Gogotsi , Y., 2008. Materials for electrochemical capacitors. *Nature Materials,* Volume 7, pp. 845-854.

Tibshirani, R., 1996. Regression Shrinkage and Selection via the Lasso. *J. R. Statist. Soc. B* , Volume 58, pp. 267-288.

Wei, J. et al., 2019. Machine Learning in Material Science. *Info Mat,* I(3).

APPENDIX

Image Processing

Reference: (Rabbani & Salehi, 2017)

MATLAB CODE

```
% Extraction of porosity and pore size distribution from SEM images

% We assume that the input SEM images are gray-scale and darker parts of
the image
% shows deeper surfaces which are considered as pore spaces

% Please cite these paper:

% Rabbani, A., & Salehi, S. (2017). Dynamic modeling of the formation
% damage and mud cake deposition using filtration theories coupled with
% SEM image processing. Journal of Natural Gas Science and Engineering, 42,
157-168.

% Ezeakacha, C. P., Rabbani, A., Salehi, S., & Ghalambor, A. (2018,
February).
% Integrated Image Processing and Computational Techniques to Characterize
Formation
% Damage. In SPE International Conference and Exhibition on Formation
Damage Control.
% Society of Petroleum Engineers.

% By Arash Rabbani
% arashrabbani.com
% rabarash@yahoo.com

clear; close all; clc;
File_Name='3a.png'; % I have put 2 sample images in the folder SEM1.jpg and
SEM2.jpg
Resolution=0.022727273; % micron/pixel % this is the spatial resolution of
the input
N=8; % Number of intensity levels in the image,
% if you think that the result porosity is overestimated, just increase
this number and vice versa, it accepts integers
A=imread(File_Name);
if ndims(A)==3; B=rgb2gray(A); end

level = multithresh(B,N);
C= imquantize(B,level);
RGB1 = label2rgb(B);
imwrite(RGB1,[File_Name(1:end-4) '_Depth Map.png']);

P=zeros(size(C));
for I=1:size(C,1)
    for J=1:size(C,2)
        if C(I,J)==1
            P(I,J)=1;
        end
    end
end
P=1-P;
P=bwmorph(P,'majority',1);
```

```
imwrite(P,[File_Name(1:end-4) '_Binary Segmentation.png']);

Conn=8;
[s1,s2]=size(P);
D=-bwdist(P,'cityblock');
B=medfilt2(D,[3 3]);
B=watershed(B,Conn);
Pr=zeros(s1,s2);

for I=1:s1
    for J=1:s2
        if P(I,J)==0 && B(I,J)~=0
            Pr(I,J)=1;
        end
    end
end
Pr=bwareaopen(Pr,9,Conn);

[Pr_L,Pr_n]=bwlabel(Pr,Conn);
RGB2 = label2rgb(Pr_L,'jet','white','shuffle');
imwrite(RGB2,[File_Name(1:end-4) '_Pore Space Segmentation.png']);
V=zeros(Pr_n,1);
for I=1:s1
    for J=1:s2
        if Pr_L(I,J)~=0
            V(Pr_L(I,J))=V(Pr_L(I,J))+1;
        end
    end
end
SP=4*pi*sum(sum(Pr))/(sum(sum(bwperim(Pr,4))))^2;
X=Resolution.*(V./pi).^.5; % Pore radius
Porosity=1-mean(P(:))
Average_Pore_radius=mean(X) % micron
Standard_Deviation_of_Pore_radius=std(X) % micron

figure;
subplot(2,3,1); imshow(A); title('Original SEM Image');
subplot(2,3,2); imshow(RGB1); title('Depth Map');
subplot(2,3,3); imshow(P); title('Binary Segmentation');
subplot(2,3,4); imshow(RGB2); title('Pore Space Segmentation');
annotation('textbox',[0 .9 .1 .1], 'String', [ 'Porosity = '
num2str(Porosity) ' (fraction)']);
subplot(2,3,5:6); hist(X,25); xlabel('Pore Radius (um)');
ylabel('Frequency'); title('Pore Size Distribution');
set(gcf, 'Position' , get(0, 'Screensize' ));
```

Correlation Matrix

PYTHON CODE

Reference: (Naik, 2020)

#importing libraries

from sklearn.datasets import load_boston

```
import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

#Loading the dataset

dataset = pd.read_excel("DataSet.xlsx",)

pd.set_option('display.max_columns', None)

dataset.head()

data= dataset.drop(['ElectMG','ElectrolyteType','Separator','PreparationMG','Power'], axis=1)

data=data[data.Energy<70]

data=data[data.Porosity<70]

X = data.drop("Energy",axis=1)   #Feature Matrix

y = data["Energy"]

# separate dataset into train and test

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(

    X,

    y,

    test_size=0.3,

    random_state=0)


X_train.shape, X_test.shape

X_train.corr()

import seaborn as sns

#Using Pearson Correlation
```

```
plt.figure(figsize=(12,10))

cor = X_train.corr()

sns.heatmap(cor, annot=True, cmap=plt.cm.Blues)

plt.show()


def correlation(dataset, threshold):

    col_corr = set()  # Set of all the names of correlated columns

    corr_matrix = dataset.corr()

    for i in range(len(corr_matrix.columns)):

        for j in range(i):

            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value

                colname = corr_matrix.columns[i]  # getting the name of column

                col_corr.add(colname)

    return col_corr

corr_features = correlation(X_train, 0.65)

len(set(corr_features))

corr_features
```

Model Training & Predictions

```
clc

clear all

opts = spreadsheetImportOptions("NumVariables", 25);


% Specify sheet and range
opts.Sheet = "Data";

opts.DataRange = "B2:Z2190";


% Specify column names and types
```

```
opts.VariableNames = ["Capacitance", "Power", "Energy", "ElectMG", "HeatT",
"SSA",   "IDIG",   "Nconcentration",   "Oconcentration",   "Pconcentration",
"Bconcentration",      "Porosity",      "PoreVolumecm3g",      "PreparationMG",
"ElectrolyteType",     "Electrolyteconcentration",     "NormalizedScanRate",
"AbsolutePotentialWindow",          "Separator",          "CurrentCollector",
"SaltAnionVolume",         "SaltCationVolume",         "SolventDipoleMoment",
"SolventVolume", "BinderConc"];

opts.VariableTypes = ["double", "double", "double", "categorical", "double",
"double",  "double",  "double",  "double",  "double",  "double",  "double",
"double",  "categorical",  "categorical",  "double",  "double",  "double",
"categorical",  "categorical",  "double",  "double",  "double",  "double",
"double"];



% Specify variable properties
opts  =  setvaropts(opts,  ["ElectMG",  "PreparationMG",  "ElectrolyteType",
"Separator", "CurrentCollector"], "EmptyFieldRule", "auto");



% Import the data
DataSet   =   readtable("D:\Kullanıcılar\canko\OneDrive\Masaüstü\Tubitak\100
Makale\DataSet.xlsx", opts, "UseExcel", false);
```

```
%Remove other response variables

Inputs = removevars(DataSet,[1,2]);
```

```
%Remove outlier data
%By changing round(control) value someone can adjust amount of data
%preserved throughout the code. This is done to decrease RMSE.
```

```
control=0;

i=0;

while round(control)~=70

control= (size(Inputs(Inputs.Energy<i,:),1)/size(Inputs,1))*100;

control;

i=i+1;

end

threshold=i;

Inputs= Inputs(Inputs.Energy<threshold,:);

Inputs= Inputs(Inputs.PoreVolumecm3g<70,:);
```

```
%Normalize numerical data
numeric= removevars(Inputs, [2,12,13,17,18]);


% normalized data except target variable to get rid of scale inconsistance
numeric{:, 2:end} = normalize(numeric{:, 2:end},'range',[0 1]);

categoric=Inputs(:,[2,12,13,17,18]);

Inputs=[numeric categoric];


Inputs   =   movevars(Inputs,'Energy','After','CurrentCollector');   %Energy
output removed at last column
```

```
%F-Test: First feature selection method


[idx,scores] = fsrftest(Inputs,"Energy");


find(isinf(scores));


bar(scores(idx))
xlabel('Predictor rank')

ylabel('Predictor importance score')

labels=Inputs.Properties.VariableNames(idx);

xticks(1:22) %Labellar arasındaki uzaklığı ayarlıyor

xticklabels(strrep(Inputs.Properties.VariableNames(idx),'_','\_'))

xtickangle(45)

set(gca,'FontSize',18)

set(gca,'YMinorTick','on')

set(gca,'linewidth',3)

Inputs(:,idx);


Inputs=removevars(Inputs, [idx(end),idx(end-1),idx(end-2)]);
```

```
% REMOVE FEATURES ACCORDİNG TO TRİAL&ERROR METHOD
Inputs=removevars(Inputs,
["HeatT","Nconcentration","Oconcentration","Porosity","PoreVolumecm3g", ...

    "SaltAnionVolume","SaltCationVolume","BinderConc","ElectrolyteType",
...
```

```
    "CurrentCollector","SolventDipoleMoment","ElectMG"]);
```

```
Data=Inputs; %Store data as categorical to use suitable models
%Dummy Coding


%Dummy_ElectMG=array2table(dummyvar(Inputs.ElectMG), ...
%
'VariableNames',{'ElectMG1','ElectMG2','ElectMG3','ElectMG4','ElectMG5','El
ectMG6','ElectMG7','ElectMG8','ElectMG9','ElectMG10','ElectMG11','ElectMG12
','ElectMG13','ElectMG14','ElectMG15'});

Dummy_PreparationMG=array2table(dummyvar(Inputs.PreparationMG), ...

"VariableNames",{'Dummy_PreparationMG1','Dummy_PreparationMG2','Dummy_Prepa
rationMG3','Dummy_PreparationMG4','Dummy_PreparationMG5','Dummy_Preparation
MG6','Dummy_PreparationMG7','Dummy_PreparationMG8','Dummy_PreparationMG9','
Dummy_PreparationMG10','Dummy_PreparationMG11'});

%Dummy_ElectrolyteType=array2table(dummyvar(Inputs.ElectrolyteType) ...

%
,"VariableNames",{'ElectrolyteType1','ElectrolyteType2','ElectrolyteType3',
});

Dummy_Separator=array2table(dummyvar(Inputs.Separator), ...

"VariableNames",{'Separator1','Separator2','Separator3','Separator4','Separ
ator5','Separator6','Separator7'});

%Dummy_CurrentCollector=array2table(dummyvar(Inputs.CurrentCollector), ...

%
"VariableNames",{'CurrentCollector1','CurrentCollector2','CurrentCollector3
','CurrentCollector4','CurrentCollector5','CurrentCollector6','CurrentColle
ctor7','CurrentCollector8','CurrentCollector9','CurrentCollector10','Curren
tCollector11'});

ResponseVar=array2table(Inputs.Energy);


%'ElectMG','ElectrolyteType','CurrentCollector',
% Dummy_ElectMG Dummy_ElectrolyteType Dummy_CurrentCollector


Inputs=removevars(Inputs, {'PreparationMG', 'Separator',  'Energy'});


Inputs=[Inputs  Dummy_PreparationMG  Dummy_Separator  ResponseVar]
Inputs.Properties.VariableNames{'Var1'} ='Energy'
```

```
%Splitting data to train,test and validation with K-Fold Cross validation
%for the dummied data

rand_num = randperm(size(Inputs,1));

%Splitting Train and Test Data 90% of data

InputsTT=table2array(Inputs(rand_num(1:round(0.9*length(rand_num))),:));



%Splitting Validation Data 10% of data
InputV                                                                       =
table2array(Inputs(rand_num(round(0.9*length(rand_num))+1:end),:));

InputValidX=InputV(:,1:size(InputV,2)-1);

InputValidY=InputV(:,size(InputV,2));



%Cross validation
c = cvpartition(size(InputsTT,1),'KFold',5); %80% of data train 20% of data
test

idx1=training(c,1);

idx2=training(c,2);

idx3=training(c,3);

idx4=training(c,4);

idx5=training(c,5);




InputTrain1 = (InputsTT(idx1,:));
InputTrainX1 = InputTrain1(:,1:size(InputTrain1,2)-1);

InputTrainY1 = InputTrain1(:,size(InputTrain1,2));

InputTest1 = (InputsTT(~idx1,:));

InputTestX1 = InputTest1(:,1:size(InputTest1,2)-1);

InputTestY1 = InputTest1(:,size(InputTest1,2));



InputTrain2 = (InputsTT(idx2,:));
InputTrainX2 = InputTrain2(:,1:size(InputTrain2,2)-1);

InputTrainY2 = InputTrain2(:,size(InputTrain2,2));

InputTest2 = (InputsTT(~idx2,:));

InputTestX2 = InputTest2(:,1:size(InputTest2,2)-1);

InputTestY2 = InputTest2(:,size(InputTest2,2));
```

```
InputTrain3 = (InputsTT(idx3,:));
InputTrainX3 = InputTrain3(:,1:size(InputTrain3,2)-1);

InputTrainY3 = InputTrain3(:,size(InputTrain3,2));

InputTest3 = (InputsTT(~idx3,:));

InputTestX3 = InputTest3(:,1:size(InputTest3,2)-1);

InputTestY3 = InputTest3(:,size(InputTest3,2));


InputTrain4 = (InputsTT(idx4,:));
InputTrainX4 = InputTrain4(:,1:size(InputTrain4,2)-1);

InputTrainY4 = InputTrain4(:,size(InputTrain4,2));

InputTest4 = (InputsTT(~idx4,:));

InputTestX4 = InputTest4(:,1:size(InputTest4,2)-1);

InputTestY4 = InputTest4(:,size(InputTest4,2));


InputTrain5 = (InputsTT(idx5,:));
InputTrainX5 = InputTrain5(:,1:size(InputTrain5,2)-1);

InputTrainY5 = InputTrain5(:,size(InputTrain5,2));

InputTest5 = (InputsTT(~idx5,:));

InputTestX5 = InputTest5(:,1:size(InputTest5,2)-1);

InputTestY5 = InputTest5(:,size(InputTest5,2));


InputTrainX=   {InputTrainX1   InputTrainX2   InputTrainX3   InputTrainX4
InputTrainX5};
InputTrainY=   {InputTrainY1   InputTrainY2   InputTrainY3   InputTrainY4
InputTrainY5};


InputTestX= {InputTestX1 InputTestX2 InputTestX3 InputTestX4 InputTestX5};
InputTestY= {InputTestY1 InputTestY2 InputTestY3 InputTestY4 InputTestY5};
```

```
%Splitting data to train,test and validation with K-Fold Cross validation
%for the catorical data included


c = cvpartition(size(Data,1),'KFold',5); %80% of data train 20% of data test
idx1=training(c,1);
```

```
idx2=training(c,2);

idx3=training(c,3);

idx4=training(c,4);

idx5=training(c,5);




dataTrain1 = (Data(idx1,:));
dataTrainX1 = dataTrain1(:,1:size(dataTrain1,2)-1);

dataTrainY1 = dataTrain1(:,size(dataTrain1,2));

dataTest1 = (Data(~idx1,:));

dataTestX1 = dataTest1(:,1:size(dataTest1,2)-1);

dataTestY1 = dataTest1(:,size(dataTest1,2));


dataTrain2 = (Data(idx2,:));
dataTrainX2 = dataTrain2(:,1:size(dataTrain2,2)-1);

dataTrainY2 = dataTrain2(:,size(dataTrain2,2));

dataTest2 = (Data(~idx2,:));

dataTestX2 = dataTest2(:,1:size(dataTest2,2)-1);

dataTestY2 = dataTest2(:,size(dataTest2,2));


dataTrain3 = (Data(idx3,:));
dataTrainX3 = dataTrain3(:,1:size(dataTrain3,2)-1);

dataTrainY3 = dataTrain3(:,size(dataTrain3,2));

dataTest3 = (Data(~idx3,:));

dataTestX3 = dataTest3(:,1:size(dataTest3,2)-1);

dataTestY3 = dataTest3(:,size(dataTest3,2));


dataTrain4 = (Data(idx4,:));
dataTrainX4 = dataTrain4(:,1:size(dataTrain4,2)-1);

dataTrainY4 = dataTrain4(:,size(dataTrain4,2));

dataTest4 = (Data(~idx4,:));

dataTestX4 = dataTest4(:,1:size(dataTest4,2)-1);

dataTestY4 = dataTest4(:,size(dataTest4,2));
```

```
dataTrain5 = (Data(idx5,:));
dataTrainX5 = dataTrain5(:,1:size(dataTrain5,2)-1);

dataTrainY5 = dataTrain5(:,size(dataTrain5,2));

dataTest5 = (Data(~idx5,:));

dataTestX5 = dataTest5(:,1:size(dataTest5,2)-1);

dataTestY5 = dataTest5(:,size(dataTest5,2));
```

```
%TUNING LASSO ALPHA
lasso_validRMSE_tune= [];



for i=1:5
    Alpha_trial=[0.1 0.25 0.5 0.75 1];

    [B
lassoReg_tune]=lasso(cell2mat(InputTrainX(1)),cell2mat(InputTrainY(1)),"Alp
ha",Alpha_trial(:,i),"CV",5);

    idxLambda1SE_tune= lassoReg_tune.Index1SE;

    coef_tune = B(:,idxLambda1SE_tune);

    coef0_tune = lassoReg_tune.Intercept(idxLambda1SE_tune);


    lasso_validPred_tune = InputValidX*coef_tune + coef0_tune;

    lasso_validRMSE_tune(:,i)           =           sqrt(mean((InputValidY-
lasso_validPred_tune).^2));

end

[RMSE_min index_alpha]=min(lasso_validRMSE_tune);

alpha_tuned=Alpha_trial(index_alpha)
```

```
%TUNİNG RİDGE k
ridge_validRMSE_tune= [];

for i=1:25

k_trial = [0.1:0.2:5]; %Ridge parametrelerini araştır.

ridge_reg_tune
=ridge(cell2mat(InputTrainY(1)),cell2mat(InputTrainX(1)),k_trial(i),0);

ridge_validPred = ridge_reg_tune(1) + InputValidX*ridge_reg_tune(2:end);

ridge_validRMSE(:,i) = sqrt(mean((InputValidY-ridge_validPred).^2));

end

[RMSE_min index_k]=min(ridge_validRMSE);
```

```
k_tuned=k_trial(index_k)
```

```
%Fit&Predict Linear Regression
for i=1:5

linearReg = fitlm(cell2mat(InputTrainX(i)),cell2mat(InputTrainY(i)));

linear_trainPred{i} = predict(linearReg,cell2mat(InputTrainX(i)));

linear_testPred{i} = predict(linearReg,cell2mat(InputTestX(i)));

linear_trainRMSE{i}          =          sqrt(mean((cell2mat(InputTrainY(i))-
cell2mat(linear_trainPred(i))).^2));

linear_testRMSE{i}           =           sqrt(mean((cell2mat(InputTestY(i))-
cell2mat(linear_testPred(i))).^2));

linear_testRMSE0{i}              =              std(cell2mat(InputTestY(i))-
mean(cell2mat(InputTestY(i))).^2);

end

avg_linear_trainRMSE=mean(cell2mat(linear_trainRMSE))

avg_linear_testRMSE=mean(cell2mat(linear_testRMSE))

avg_linear_testRMSE0=mean(cell2mat(linear_testRMSE0));

r_sq=1-(avg_linear_testRMSE/avg_linear_testRMSE0)
```

```
% Plot for Linear Regression
for i=1:5

plot(cell2mat(InputTrainY(i)),cell2mat(linear_trainPred(i)),'o','LineWidth'
,2,'MarkerSize',15)

hold on

plot(cell2mat(InputTestY(i)),cell2mat(linear_testPred(i)),'o','LineWidth',2
,'MarkerSize',15)

hold off

end

legend('train','test');

xlabel('Actual values');

ylabel ('Predictions');

title('Linear Regression')

set(gca,'FontSize',33)

set(gca,'XMinorTick','on','YMinorTick','on')

set(gca,'TickLength',[0.02,0.05])

set(gca,'linewidth',3)
```

```
%Lasso
%To find lambda min

[P,
lassoRegres]=lasso(cell2mat(InputTrainX(1)),cell2mat(InputTrainY(1)),'Alpha
',alpha_tuned,'CV',5);

lassoPlot(P,lassoRegres,'PlotType',"CV")

legend('show')

set(gca,'FontSize',33)

set(gca,'TickLength',[0.02,0.05])

set(gca,'linewidth',3)
```

```
%Lambda min used and model is trained
minLambda=lassoRegres.LambdaMinMSE; %Değiştir DYNAMİC yap

[A,lassoReg]=lasso(cell2mat(InputTrainX(1)),cell2mat(InputTrainY(1)),'Lambd
a',minLambda,'Alpha',alpha_tuned,'CV',5);

idxLambda1SE = lassoReg.Index1SE;

coef = A(:,idxLambda1SE);

coef0 = lassoReg.Intercept(idxLambda1SE);
```

```
%Predictions and test errors for lasso regresion
for i=1:5

lasso_trainPred{i} = cell2mat(InputTrainX(i))*coef + coef0;

lasso_testPred{i} = cell2mat(InputTestX(i))*coef + coef0;

lasso_trainRMSE{i}           =          sqrt(mean((cell2mat(InputTrainY(i))-
cell2mat(lasso_trainPred(i))).^2));

lasso_testRMSE{i}            =           sqrt(mean((cell2mat(InputTestY(i))-
cell2mat(lasso_testPred(i))).^2));

lasso_testRMSE0{i}               =              std(cell2mat(InputTestY(i))-
mean(cell2mat(InputTestY(i))).^2);

end

avg_lasso_trainRMSE=mean(cell2mat(lasso_trainRMSE))

avg_lasso_testRMSE=mean(cell2mat(lasso_testRMSE))

avg_lasso_testRMSE0=mean(cell2mat(lasso_testRMSE0));

r_sq=1-(avg_lasso_testRMSE/avg_lasso_testRMSE0)
```

```
% Plot for Lasso Regression
for i=1:5

plot(cell2mat(InputTrainY(i)),cell2mat(lasso_trainPred(i)),'o','LineWidth',
2,'MarkerSize',15)

hold on

plot(cell2mat(InputTestY(i)),cell2mat(lasso_testPred(i)),'o','LineWidth',2,
'MarkerSize',15)

hold off

end

legend('train','test');

xlabel('Actual values');

ylabel ('Predictions'); title('Lasso Regression')

set(gca,'FontSize',33)

set(gca,'XMinorTick','on','YMinorTick','on')

set(gca,'TickLength',[0.02,0.05])

set(gca,'linewidth',3)
```

```
%Fit&Predict Regression Tree
RegressionTree = fitrtree(dataTrainX1,dataTrainY1);

regtree_trainPred1 = predict(RegressionTree,dataTrainX1);

regtree_testPred1 = predict(RegressionTree,dataTestX1);

regtree_trainRMSE1          =          sqrt(mean((table2array(dataTrainY1)-
regtree_trainPred1).^2));

regtree_testRMSE1           =           sqrt(mean((table2array(dataTestY1)-
regtree_testPred1).^2));

regtree_testRMSE01             =             std((table2array(dataTestY1)-
mean(table2array(dataTestY1))).^2);



RegressionTree = fitrtree(dataTrainX2,dataTrainY2);
regtree_trainPred2 = predict(RegressionTree,dataTrainX2);

regtree_testPred2 = predict(RegressionTree,dataTestX2);

regtree_trainRMSE2          =          sqrt(mean((table2array(dataTrainY2)-
regtree_trainPred2).^2));

regtree_testRMSE2           =           sqrt(mean((table2array(dataTestY2)-
regtree_testPred2).^2));
```

```
regtree_testRMSE02             =            std((table2array(dataTestY2)-
mean(table2array(dataTestY2))).^2);



RegressionTree = fitrtree(dataTrainX3,dataTrainY3);
regtree_trainPred3 = predict(RegressionTree,dataTrainX3);

regtree_testPred3 = predict(RegressionTree,dataTestX3);

regtree_trainRMSE3          =          sqrt(mean((table2array(dataTrainY3)-
regtree_trainPred3).^2));

regtree_testRMSE3           =           sqrt(mean((table2array(dataTestY3)-
regtree_testPred3).^2));

regtree_testRMSE03             =            std((table2array(dataTestY3)-
mean(table2array(dataTestY3))).^2);



RegressionTree = fitrtree(dataTrainX4,dataTrainY4);
regtree_trainPred4 = predict(RegressionTree,dataTrainX4);

regtree_testPred4 = predict(RegressionTree,dataTestX4);

regtree_trainRMSE4          =          sqrt(mean((table2array(dataTrainY4)-
regtree_trainPred4).^2));

regtree_testRMSE4           =           sqrt(mean((table2array(dataTestY4)-
regtree_testPred4).^2));

regtree_testRMSE04             =            std((table2array(dataTestY4)-
mean(table2array(dataTestY4))).^2);



RegressionTree = fitrtree(dataTrainX5,dataTrainY5);
regtree_trainPred5 = predict(RegressionTree,dataTrainX5);

regtree_testPred5 = predict(RegressionTree,dataTestX5);

regtree_trainRMSE5          =          sqrt(mean((table2array(dataTrainY5)-
regtree_trainPred5).^2));

regtree_testRMSE5           =           sqrt(mean((table2array(dataTestY5)-
regtree_testPred5).^2));

regtree_testRMSE05             =            std((table2array(dataTestY5)-
mean(table2array(dataTestY5))).^2);



avg_regtree_trainRMSE=(regtree_trainRMSE1+regtree_trainRMSE2+regtree_trainR
MSE3+regtree_trainRMSE4+regtree_trainRMSE5 )/5
avg_regtree_testRMSE=(regtree_testRMSE1+regtree_testRMSE2+regtree_testRMSE3
+regtree_testRMSE4+regtree_testRMSE5)/5

avg_regtree_testRMSE0=(regtree_testRMSE01+regtree_testRMSE02+regtree_testRM
SE03+regtree_testRMSE04+regtree_testRMSE05)/5;
```

```
r_sq=1-(avg_regtree_testRMSE/avg_regtree_testRMSE0)
```

```
% Plot for Regression Tree
plot(table2array(dataTrainY1),regtree_trainPred1,'o','LineWidth',2,'MarkerS
ize',15)

hold on

plot(table2array(dataTestY1),regtree_testPred1,'o','LineWidth',2,'MarkerSiz
e',15)

hold off

legend('train','test');

xlabel('Actual values');

ylabel ('Predictions'); title('Regression Tree')

set(gca,'FontSize',33)

set(gca,'XMinorTick','on','YMinorTick','on')

set(gca,'TickLength',[0.02,0.05])

set(gca,'linewidth',3)
```

```
%   Ridge model training&predictions
k = k_tuned;

for i=1:5

ridge_reg =ridge(cell2mat(InputTrainY(i)),cell2mat(InputTrainX(i)),k,0);

ridge_trainPred{i}               =              ridge_reg(1)              +
cell2mat(InputTrainX(i))*ridge_reg(2:end);

ridge_testPred{i} = ridge_reg(1) + cell2mat(InputTestX(i))*ridge_reg(2:end);

ridge_trainRMSE{i}           =          sqrt(mean((cell2mat(InputTrainY(i))-
cell2mat(ridge_trainPred(i))).^2));

ridge_testRMSE{i}            =          sqrt(mean((cell2mat(InputTestY(i))-
cell2mat(ridge_testPred(i))).^2));

ridge_testRMSE0{i}               =              std(cell2mat(InputTestY(i))-
mean(cell2mat(InputTestY(i))).^2);

end

avg_ridge_trainRMSE=mean(cell2mat(ridge_trainRMSE))

avg_ridge_testRMSE=mean(cell2mat(ridge_testRMSE))

avg_ridge_testRMSE0=mean(cell2mat(ridge_testRMSE0));
```

```
r_sq=1-(avg_ridge_testRMSE/avg_ridge_testRMSE0)
```

```
%plot ridge regression
for i=1:5

plot(cell2mat(InputTrainY(i)),cell2mat(ridge_trainPred(i)),'o','LineWidth',
2,'MarkerSize',15)

hold on

plot(cell2mat(InputTestY(i)),cell2mat(ridge_testPred(i)),'o','LineWidth',2,
'MarkerSize',15)

hold off

end

legend('train','test');

xlabel('Actual values');

ylabel ('Predictions'); title('Ridge Regression')

set(gca,'FontSize',33)

set(gca,'XMinorTick','on','YMinorTick','on')

set(gca,'TickLength',[0.02,0.05])

set(gca,'linewidth',3)
```

```
%Neural Network Modelling
hiddenLayerSize = 25

trainFcn = 'trainlm'

net = fitnet(hiddenLayerSize,trainFcn)

net.divideParam.trainRatio = 72/100;

net.divideParam.valRatio = 10/100;

net.divideParam.testRatio = 18/100;


rng('default')
[NeuralNet,     err]     =     train(net,(table2array(Inputs(:,1:end-1)))',
(Inputs.Energy)')
```