MACHINE LEARNING

PROJECT REPORT ON

REMAINING USEFUL LIFE ESTIMATION FOR LI-ION BATTERIES

BASED ON SUPPORT VECTOR MACHINE

BY

Suvarna Jadhav 171081031
Hritika Panjabi 171081062
Payal Bathija 171081064
Apurva Mahajan 171081081
Dhanashree Bhandare 181081908

TY BTECH IT

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

DATE OF PROJECT: 05/04/2020

PRE-REQUISITES

INSTALLING ALL USEFUL AND REQUIRED LIBRARIES:

1) INSTALL PYTHON IN YOUR SYSTEM USING OFFICIAL PYTHON LINK
   https://www.python.org/downloads/

   USE FOLLOWING GUIDELINE TO SETUP:
   https://realpython.com/installing-python/

2) INSTALL PANDAS USING pip install pandas
3) INSTALL TENSORFLOW USING pip install tensorflow
4) INSTALL KERAS USING pip install keras
5) INSTALL NUMPY USING pip install numpy
6) INSTALL SCIPY USING pip install scipy
7) INSTALL MATPLOTLIB USING pip install matplotlib

HARDWARE REQUIREMENTS:

RAM : 8GB
PROCESSOR : i3 and more

IMPLEMENTATION:

We will estimate the Remaining Useful Life of Li-ion batteries with help of a dataset using **SVM** and improve results by using **Neural Networks**

//converting dataset into raw data (text format)

Converting .mat file to .csv :

```
import scipy.io
import numpy as np

data = scipy.io.loadmat("B006.mat")

for i in data:
    if '__' not in i and 'readme' not in i:
        np.savetxt(("Batt/"+i+".csv"),data[i],delimiter=',')
```

IMPORTING ALL THE REQUIRED LIBRARIES

In [1]:

```
import pandas as pd
import numpy as np
import itertools
import matplotlib.pyplot as plt
import random
import os

from scipy.spatial.distance import pdist, squareform
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import MinMaxScaler, StandardScaler

import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.utils import *
from tensorflow.keras.callbacks import *
```

UNDERSTANDING THE RAW DATA

In [2]:

```
### LOAD DATA ###
train_df = pd.read_csv('./data.txt', sep=" ", header=None)
train_dycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                 's15', 's16', 's17', 's18', 's19', 's20', 's21']
print('#id:',lef.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
train_df.columns = ['id', 'cn(train_df.id.unique()))
train_df = train_df.sort_values(['id','cycle'])
print(train_df.shape)
train_df.head(3)
```

```
#id: 100
(20631, 26)
```

Out[2]:

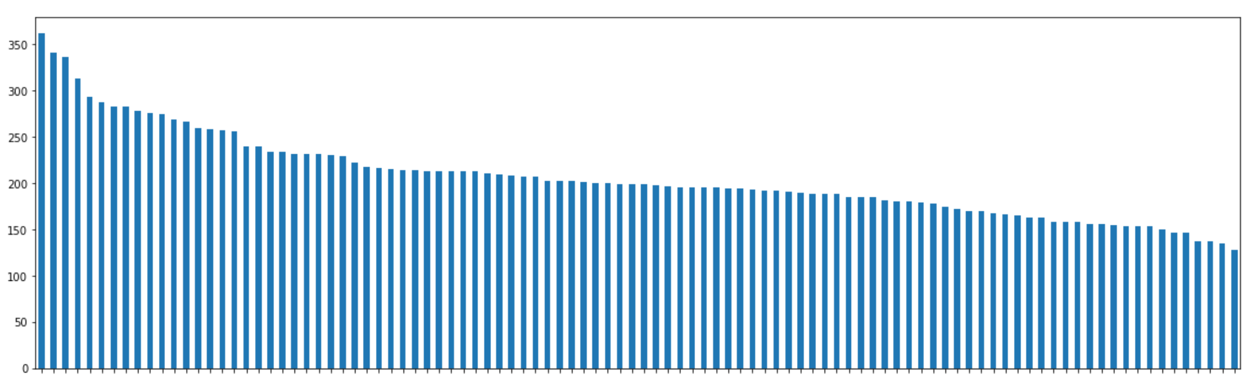
	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442

3 rows × 26 columns

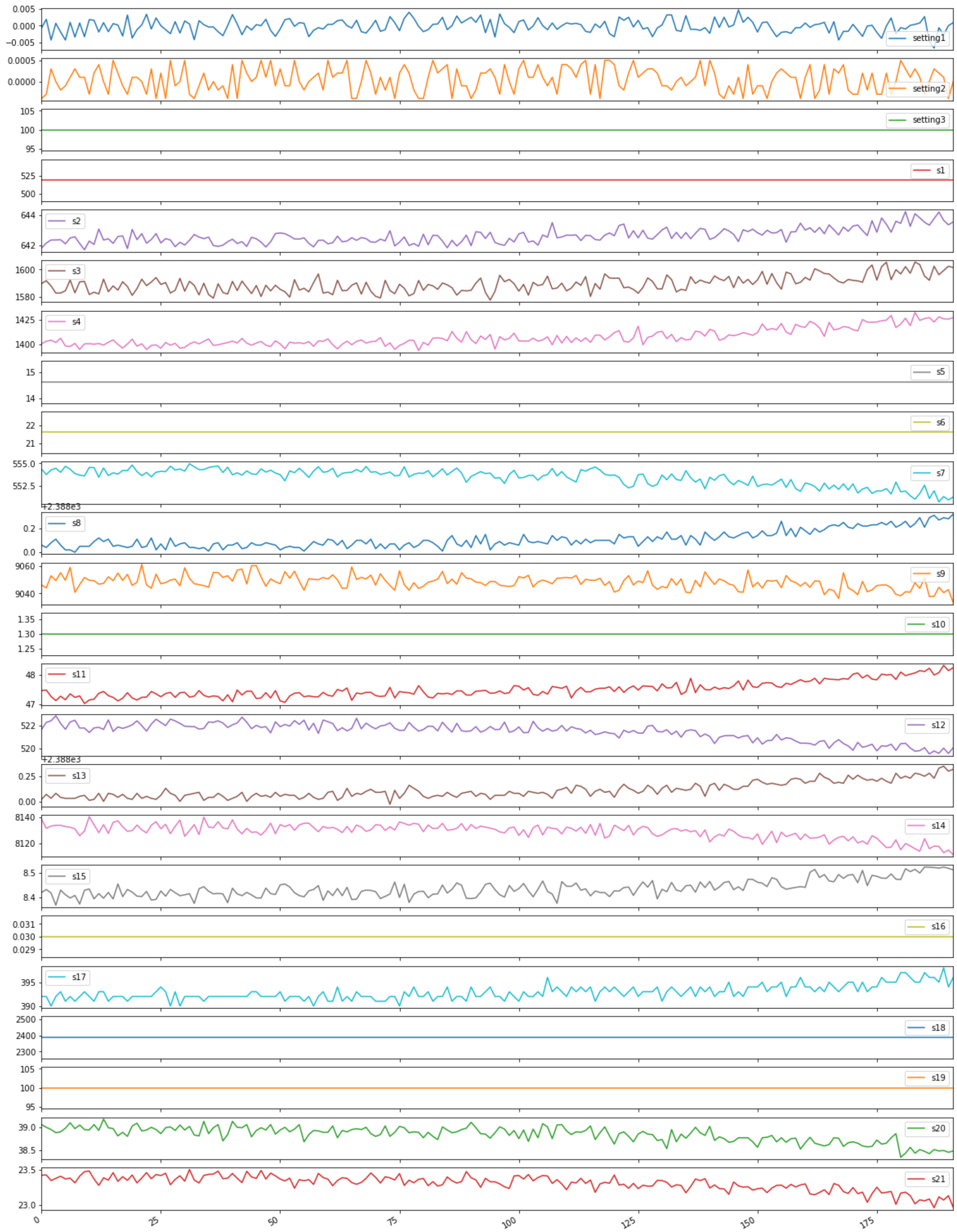
In [3]:

```
### PLOT TRAINING FREQ ###
plt.figure(figsize=(20,6))
train_df.id.value_counts().plot.bar()
print("mean working time:", train_df.id.value_counts().mean())
print("max working time:", train_df.id.value_counts().max())
print("min working time:", train_df.id.value_counts().min())
```

mean working time: 206.31
max working time: 362
min working time: 128



In [4]:
```
### plotting data for battery ID ###
battery_id = train_df[train_df['id'] == 1]

ax1 = battery_id[train_df.columns[2:]].plot(subplots=True, sharex=True,
figsize=(20,30))
```



In [5]:
```
### LOAD TEST ###
test_df = pd.read_csv('./test.txt', sep=" ", header=None)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
test_df.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                   's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                   's15', 's16', 's17', 's18', 's19', 's20', 's21']
print('#id:',len(test_df.id.unique()))
print(test_df.shape)
test_df.head(3)
```

#id: 100
(13096, 26)

Out[5]:

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	521.97	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166

3 rows × 26 columns

In [7]:
```
### CALCULATE RUL ###
train_df['RUL']=train_df.groupby(['id'])['cycle'].transform(max)-train_df['cycle']
train_df.RUL[0:10]
```

Out[7]:
```
0    191
1    190
2    189
3    188
4    187
5    186
6    185
7    184
8    183
9    182
Name: RUL, dtype: int64
```

In [8]:
```
### ADD NEW LABEL ###
w1 = 45
w0 = 15
train_df['label1'] = np.where(train_df['RUL'] <= w1, 1, 0 )
train_df['label2'] = train_df['label1']
train_df.loc[train_df['RUL'] <= w0, 'label2'] = 2
```

NORMALIZING THE TRAINING DATA:

In [9]:
```
### SCALE TRAIN DATA ###
def scale(df):
    #return (df - df.mean())/df.std()
    return (df - df.min())/(df.max()-df.min())

for col in train_df.columns:
    if col[0] == 's':
        train_df[col] = scale(train_df[col])
#    elif col == 'cycle':
#        train_df['cycle_norm'] = scale(train_df[col])

train_df = train_df.dropna(axis=1)
train_df.head()
```

Out[9]:

	id	cycle	setting1	setting2	s2	s3	s4	s6	s7	s8	...	s12	s13	s14	s15	s17	s20	s21	RUL	label1	label2
0	1	1	0.459770	0.166667	0.183735	0.406802	0.309757	1.0	0.726248	0.242424	...	0.633262	0.205882	0.199608	0.363986	0.333333	0.713178	0.724662	191	0	0
1	1	2	0.609195	0.250000	0.283133	0.453019	0.352633	1.0	0.628019	0.212121	...	0.765458	0.279412	0.162813	0.411312	0.333333	0.666667	0.731014	190	0	0
2	1	3	0.252874	0.750000	0.343373	0.369523	0.370527	1.0	0.710145	0.272727	...	0.795309	0.220588	0.171793	0.357445	0.166667	0.627907	0.621375	189	0	0
3	1	4	0.540230	0.500000	0.343373	0.256159	0.331195	1.0	0.740741	0.318182	...	0.889126	0.294118	0.174889	0.166603	0.333333	0.573643	0.662386	188	0	0
4	1	5	0.390805	0.333333	0.349398	0.257467	0.404625	1.0	0.668277	0.242424	...	0.746269	0.235294	0.174734	0.402078	0.416667	0.589147	0.704502	187	0	0

5 rows × 22 columns

In [10]:

```
### CALCULATE RUL TEST ###
truth_df['max'] = test_df.groupby('id')['cycle'].max() + truth_df['more']
test_df['RUL'] = [truth_df['max'][i] for i in test_df.id] - test_df['cycle']
```

In [11]:

```
### ADD NEW LABEL TEST ###
test_df['label1'] = np.where(test_df['RUL'] <= w1, 1, 0 )
test_df['label2'] = test_df['label1']
test_df.loc[test_df['RUL'] <= w0, 'label2'] = 2
```

In [12]:
```
### SCALE TEST DATA ###

for col in test_df.columns:
    if col[0] == 's':
        test_df[col] = scale(test_df[col])
#     elif col == 'cycle':
#         test_df['cycle_norm'] = scale(test_df[col])

test_df = test_df.dropna(axis=1)
test_df.head()
```

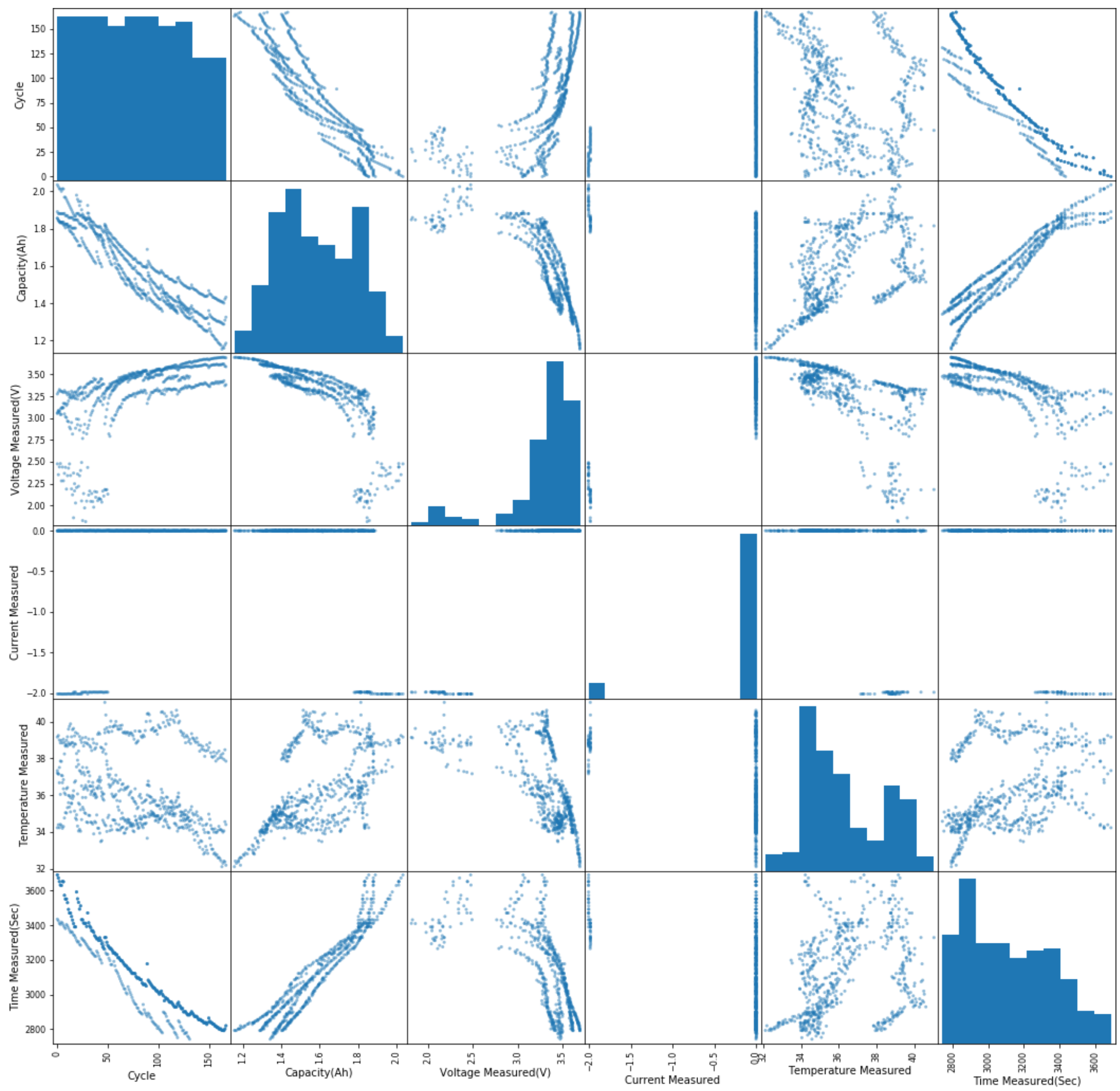
Out[12]:

	id	cycle	setting1	setting2	s2	s3	s4	s6	s7	s8	...	s12	s13	s14	s15	s17	s20	s21	RUL	label1	label2
0	1	1	0.65625	0.692308	0.596215	0.421968	0.282214	1.0	0.608871	0.365854	...	0.534247	0.325581	0.152259	0.347076	0.375	0.500000	0.620099	142	0	0
1	1	2	0.34375	0.230769	0.182965	0.504025	0.225240	1.0	0.800403	0.292683	...	0.634703	0.395349	0.277907	0.227709	0.500	0.645455	0.645718	141	0	0
2	1	3	0.53125	0.538462	0.419558	0.464814	0.346130	1.0	0.651210	0.390244	...	0.591324	0.325581	0.192892	0.533557	0.500	0.700000	0.681104	140	0	0
3	1	4	0.77500	0.461538	0.413249	0.391587	0.449867	1.0	0.643145	0.341463	...	0.456621	0.372093	0.217896	0.282359	0.250	0.627273	0.620382	139	0	0
4	1	5	0.60000	0.461538	0.435331	0.471306	0.357974	1.0	0.661290	0.292683	...	0.632420	0.325581	0.187891	0.337009	0.125	0.618182	0.676008	138	0	0

5 rows × 22 columns

Scatter plot matrix shows relation between different feature to one another.

In [13]:
```
# Create scatter plot matrix
from pandas.plotting import
scatter_matrix scatter_matrix(df, figsize =
(18,18)) plt.show()
```



GENERATING SEQUENCE

In [18]:
```
sequence_length = 50

def gen_sequence(id_df, seq_length, seq_cols):

    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]
    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length,
num_elements)):
        yield data_matrix[start:stop, :]

def gen_labels(id_df, seq_length, label):

    data_matrix = id_df[label].values
    num_elements = data_matrix.shape[0]

    return data_matrix[seq_length:num_elements, :]
```

In [19]:
```
### SEQUENCE COL: COLUMNS TO CONSIDER ###
sequence_cols = []
for col in train_df.columns:
    if col[0] == 's':
        sequence_cols.append(col)
#sequence_cols.append('cycle_norm')
print(sequence_cols)
```

```
['setting1', 'setting2', 's2', 's3', 's4', 's6', 's7', 's8', 's9', 's11', 's12', 's13', 's14', 's15', 's17', 's20', 's21']
```

In [20]:
```
### GENERATE X TRAIN TEST ###
x_train, x_test = [], []
for battery_id in train_df.id.unique():
    for sequence in gen_sequence(train_df[train_df.id==battery_id], sequence_length,
sequence_cols):
        x_train.append(sequence)
    for sequence in gen_sequence(test_df[test_df.id==battery_id], sequence_length,
sequence_cols):
        x_test.append(sequence)
x_train = np.asarray(x_train)
x_test = np.asarray(x_test)

print("X_Train shape:", x_train.shape)
print("X_Test shape:", x_test.shape)
```

```
X_Train shape: (15631, 50, 17)
```

X_Test shape: (8162, 50, 17)

In [21]:
```
### GENERATE Y TRAIN TEST ###
y_train, y_test = [], []
for battery_id in train_df.id.unique():
    for label in gen_labels(train_df[train_df.id==battery_id], sequence_length, ['label2']
):
        y_train.append(label)
    for label in gen_labels(test_df[test_df.id==battery_id], sequence_length, ['label2']):
        y_test.append(label)

y_train = np.asarray(y_train).reshape(-1,1)
y_test = np.asarray(y_test).reshape(-1,1)


print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

y_train shape: (15631, 1)
y_test shape: (8162, 1)

In [22]:
```
### ENCODE LABEL ###
y_train = to_categorical(y_train)
print(y_train.shape)

y_test = to_categorical(y_test)
print(y_test.shape)
```

(15631, 3)
(8162, 3)

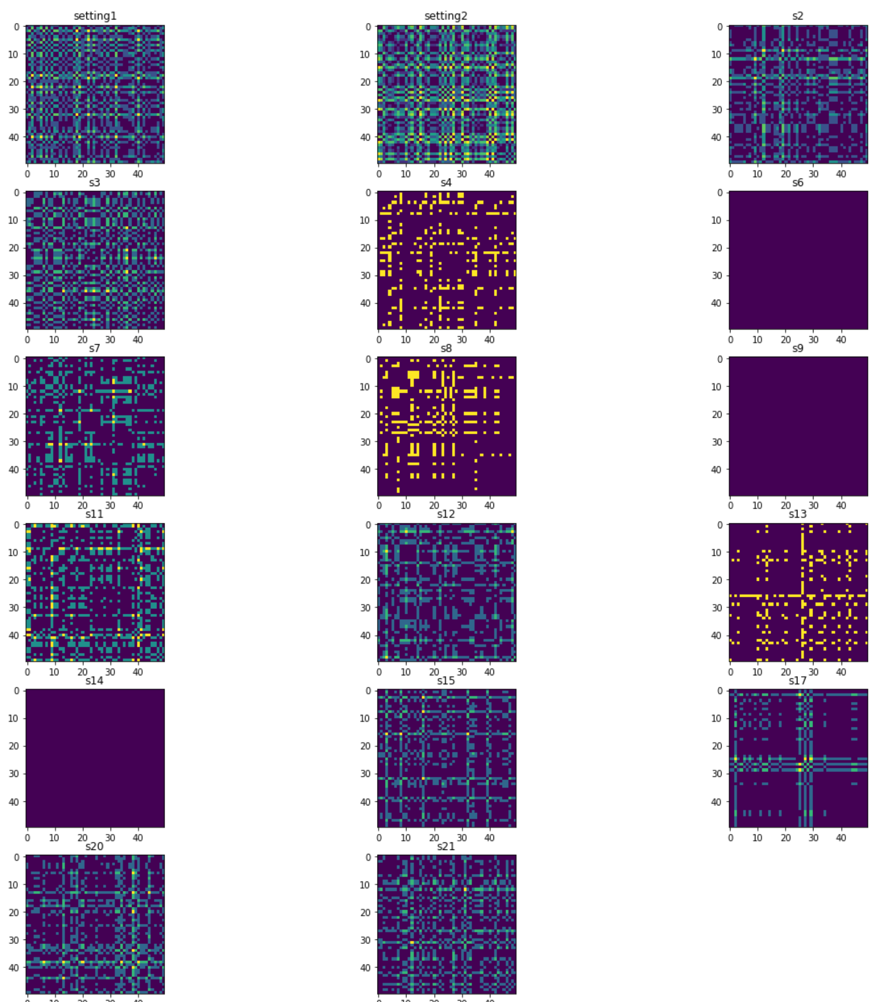
CONVERTING TIME INTO IMAGES

In [23]:

```
def rec_plot(s, eps=0.10, steps=10):
    d = pdist(s[:,None])
    d = np.floor(d/eps)
    d[d>steps] = steps
    Z = squareform(d)
    return Z
```

In [24]:

```
plt.figure(figsize=(20,20))
for i in range(0,17):

    plt.subplot(6, 3, i+1)
    rec = rec_plot(x_train[0,:,i])
    plt.imshow(rec)
    plt.title(sequence_cols[i])
plt.show()
```



In [25]:
```
### TRANSFORM X TRAIN TEST IN IMAGES ###
x_train_img = np.apply_along_axis(rec_plot, 1, x_train).astype('float16')
print(x_train_img.shape)

x_test_img = np.apply_along_axis(rec_plot, 1, x_test).astype('float16')
print(x_test_img.shape)
```

```
(15631, 50, 50, 17)
(8162, 50, 50, 17)
```

SVM MODELLING:

In [30]:
```
from sklearn.svm import SVR

model2 = SVR(kernel='',gamma='auto')

model2.fit(x_train,y_train)
```

Out[30]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',

kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [31]:
```
model2.score(x_testl,y_test)
```

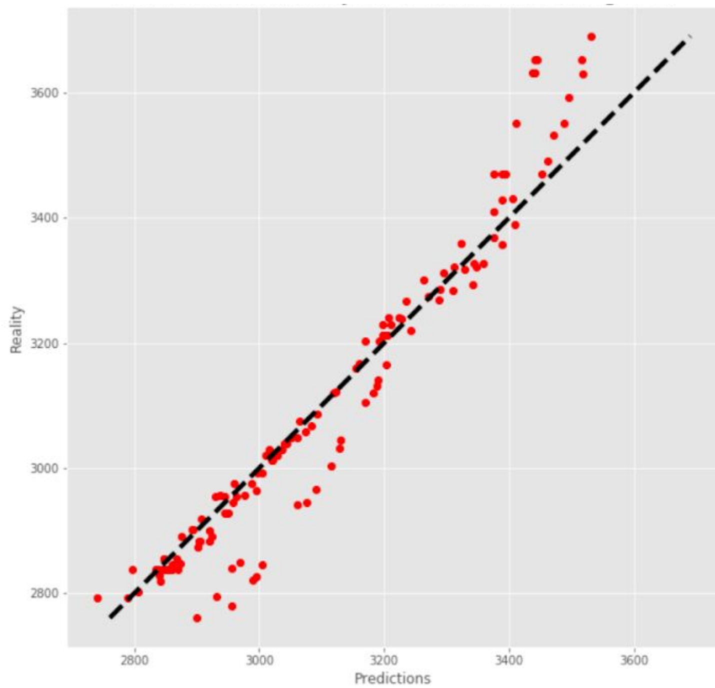
Out[31]:
```
0.9257831840874583
```

In [32]:
```
predictions2 = model2.predict(x_test)

score2 = mean_absolute_error(y_test,predictions2) score2
```

Out[32]:
```
44.023475796925815
```

Out[33]:
```
#plt.style.use(ggplot')

matplotlib.rc('xtick', labelsize=10)

matplotlib.rc('ytick', labelsize=10)

fig, ax = plt.subplots(figsize=(10, 10))

plt.plot(predictions2, y_test, 'ro')

plt.xlabel('Predictions', fontsize = 12)

plt.ylabel('Reality', fontsize=12)
```

```
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣ ↩→lw=4)

plt.show()
```



RESULT USING SVM

// Improving Test Results using Neural network

CREATING A NN MODEL

Using dense neural network, following is the implementation :

In [34]:

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 17)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 32)        4928
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 19, 19, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 9, 9, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 5184)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1327360
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 1,397,731
Trainable params: 1,397,731
Non-trainable params: 0
_________________________________________________________________
None
```

In [35]:
```
### SET SEED ###
tf.random.set_seed(33)
os.environ['PYTHONHASHSEED'] = str(33)
np.random.seed(33)
random.seed(33)

session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
sess = tf.compat.v1.Session(
    graph=tf.compat.v1.get_default_graph(),
    config=session_conf
)
tf.compat.v1.keras.backend.set_session(sess)


es = EarlyStopping(monitor='val_accuracy', mode='auto',
restore_best_weights=True, verbose=1, patience=6)

model.fit(x_train_img, y_train, batch_size=512, epochs=25, callbacks=[es],
        validation_split=0.2, verbose=2)
```

```
Train on 12504 samples, validate on 3127 samples
Epoch 1/25
12504/12504 - 74s - loss: 1.0244 - accuracy: 0.6512 - val_loss: 0.6874 - val_accuracy:
0.7352
Epoch 2/25
12504/12504 - 70s - loss: 0.7044 - accuracy: 0.6983 - val_loss: 0.6086 - val_accuracy:
0.7352
Epoch 3/25
12504/12504 - 72s - loss: 0.4659 - accuracy: 0.7912 - val_loss: 0.3295 - val_accuracy:
0.8596
Epoch 4/25
12504/12504 - 74s - loss: 0.3006 - accuracy: 0.8717 - val_loss: 0.2897 - val_accuracy:
0.8826
Epoch 5/25
12504/12504 - 78s - loss: 0.2706 - accuracy: 0.8820 - val_loss: 0.2739 - val_accuracy:
0.8775
Epoch 6/25
12504/12504 - 80s - loss: 0.2451 - accuracy: 0.8919 - val_loss: 0.2590 - val_accuracy:
0.8887
Epoch 7/25
```

```
12504/12504 - 74s - loss: 0.2335 - accuracy: 0.8990 - val_loss: 0.2685 - val_accuracy: 0.8772
Epoch 8/25
12504/12504 - 74s - loss: 0.2247 - accuracy: 0.9005 - val_loss: 0.2543 - val_accuracy: 0.8973
Epoch 9/25
12504/12504 - 75s - loss: 0.2174 - accuracy: 0.9037 - val_loss: 0.2493 - val_accuracy: 0.8900
Epoch 10/25
12504/12504 - 70s - loss: 0.2083 - accuracy: 0.9075 - val_loss: 0.2450 - val_accuracy: 0.8977
Epoch 11/25
12504/12504 - 71s - loss: 0.1917 - accuracy: 0.9153 - val_loss: 0.2725 - val_accuracy: 0.8954
Epoch 12/25
12504/12504 - 72s - loss: 0.1850 - accuracy: 0.9192 - val_loss: 0.2758 - val_accuracy: 0.8967
Epoch 13/25
12504/12504 - 72s - loss: 0.1741 - accuracy: 0.9255 - val_loss: 0.2628 - val_accuracy: 0.8951
Epoch 14/25
12504/12504 - 71s - loss: 0.1532 - accuracy: 0.9360 - val_loss: 0.2877 - val_accuracy: 0.8801
Epoch 15/25
12504/12504 - 75s - loss: 0.1315 - accuracy: 0.9432 - val_loss: 0.3285 - val_accuracy: 0.8660
Epoch 16/25
Restoring model weights from the end of the best epoch.
12504/12504 - 75s - loss: 0.1187 - accuracy: 0.9522 - val_loss: 0.3118 - val_accuracy: 0.8871
Epoch 00016: early stopping
```

Out[35]: `<tensorflow.python.keras.callbacks.History at 0x1b9f238e320>`

In [36]:

```
model.evaluate(x_test_img, y_test, verbose=2)
```

```
8162/1 - 11s - loss: 0.4127 - accuracy: 0.7857
```

Out[36]: `[0.5366564413700654, 0.78571427]`

//Viewing the confusion matrix:

In [37]:
```
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Blues):

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title, fontsize=25)
        #plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=90, fontsize=15)
        plt.yticks(tick_marks, classes, fontsize=15)

        fmt = '.2f'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black", fontsize = 14)

    plt.ylabel('True label', fontsize=20)
    plt.xlabel('Predicted label', fontsize=20)
```

CLASSIFICATION REPORT:

// report of all 3 cells

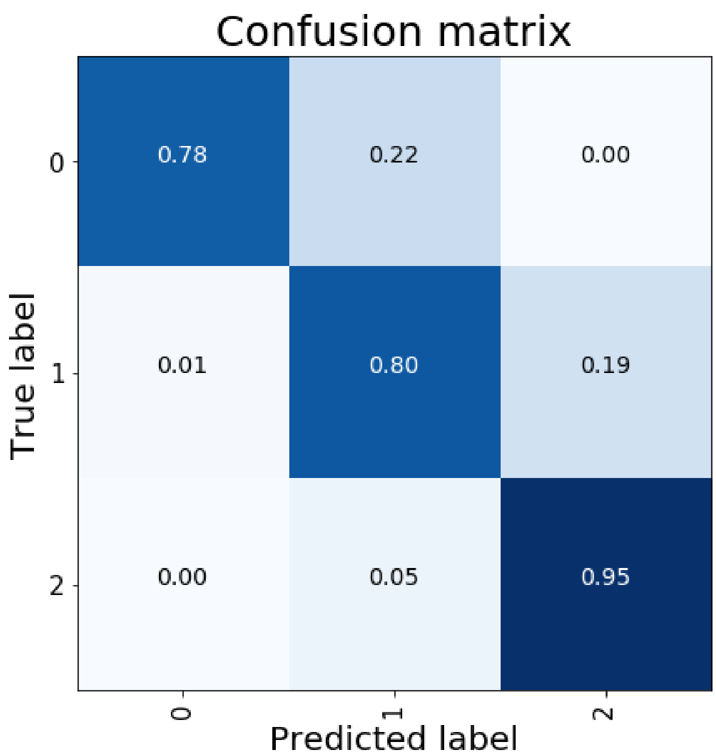
In [38]:
```
print(classification_report(np.where(y_test != 0)[1],
        model.predict_classes(x_test_img)))
```

	precision	recall	score	support
0	1.00	0.78	0.88	7426
1	0.25	0.80	0.38	676
2	0.31	0.95	0.47	60
accuracy			0.79	8162
macro avg	0.52	0.84	0.58	8162
weighted avg	0.93	0.79	0.83	8162

CONFUSION MATRIX

//creating confusion matrix to understand where mistakes are being made

In [39]:
```
cnf_matrix = confusion_matrix
(np.where(y_test !=0)[1],model.predict_classes(x_test_img))
plt.figure(figsize=(7,7))
plot_confusion_matrix(cnf_matrix, classes=np.unique(np.where(y_test != 0)[1]),
title="Confusion matrix")
plt.show()
```



In [ ]:

List of techniques which improve Neural network's performance over time that helped it to beat SVM with larger dataset:

1. Backpropagation
2. Number of hidden layers and neurons per hidden layer:
3. Activation functions

ANALYSIS OF THE PROJECT:

SVM gave 92% accuracy
While NN gave only 79% accuracy

- SVM prevented overfitting of data
- SVM typically only allowed a single transformation. Neural networks allow hundreds of layers.
- NN's accuracy increased with increase in number of hidden layers, however after a point the accuracy was almost constant but computation was taking alot of time. (6 layers in our implementation)

CONCLUSION:

SVMs work better for smaller dataset, NN is more accurate for larger dataset as Accuracy of SVM decreased with increase in the dataset.