Tennesse Eastman Falut Detection

Source: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1&version=1.0 (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1&version=1.0)

References:

1. https://drive.google.com/file/d/0B4_tSB591fQ1dmFsN0hKbjRFcVJOdVV1QTcwMjNWWnZORGh3/view (https://drive.google.com/file/d/0B4_tSB591fQ1dmFsN0hKbjRFcVJOdVV1QTcwMjNWWnZORGh3/view),
2. https://drive.google.com/file/d/1K9lw_Yh6KIE3E7QT6tQeMPSveEPvO0QT/view (https://drive.google.com/file/d/1K9lw_Yh6KIE3E7QT6tQeMPSveEPvO0QT/view)

In [1]:
```
import pandas as pd
```

In [2]:
```
import pyreadr
```

In [3]:
```
from __future__ import print_function
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D,Conv2D, MaxPooling1D,MaxPooling2D,LSTM
from keras import backend as K
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```

In [5]:
```
Faulty_Training_r = pyreadr.read_r('TEP_Faulty_Training.RData')
Faulty_Training = Faulty_Training_r['faulty_training']
del Faulty_Training_r
FaultFree_Training_r = pyreadr.read_r('TEP_FaultFree_Training.RData')
FaultFree_Training = FaultFree_Training_r['fault_free_training']
del FaultFree_Training_r
```

In [13]:
```
train_df = pd.concat([FaultFree_Training, Faulty_Training])
Y = train_df['faultNumber']
Y_train = train_df['faultNumber']
```

In [14]:
```
Faulty_Testing_r = pyreadr.read_r('TEP_Faulty_Testing.RData')
Faulty_Testing = Faulty_Testing_r['faulty_testing']
del Faulty_Testing_r
FaultFree_Testing_r = pyreadr.read_r('TEP_FaultFree_Testing.RData')
FaultFree_Testing = FaultFree_Testing_r['fault_free_testing']
del FaultFree_Testing_r

test_df = pd.concat([FaultFree_Testing, Faulty_Testing])
Y_test = test_df['faultNumber']
```

In [15]:
```
print(train_df.shape)
print(test_df.shape)
```

```
(5250000, 55)
(10080000, 55)
```

In [18]:
```
train_df.head()
```

Out[18]:

	faultNumber	simulationRun	sample	xmeas_1	xmeas_2	xmeas_3	xmeas_4	xmeas_5	xmeas_6	xmeas_7
0	0.0	1.0	1	0.25038	3674.0	4529.0	9.2320	26.889	42.402	2704.3
1	0.0	1.0	2	0.25109	3659.4	4556.6	9.4264	26.721	42.576	2705.0
2	0.0	1.0	3	0.25038	3660.3	4477.8	9.4426	26.875	42.070	2706.2
3	0.0	1.0	4	0.24977	3661.3	4512.1	9.4776	26.758	42.063	2707.2
4	0.0	1.0	5	0.29405	3679.0	4497.0	9.3381	26.889	42.650	2705.1

5 rows × 55 columns

In [9]:
```
Y_train.value_counts()
```

Out[9]:
```
1.0     250000
10.0    250000
2.0     250000
3.0     250000
4.0     250000
5.0     250000
6.0     250000
7.0     250000
8.0     250000
9.0     250000
11.0    250000
20.0    250000
12.0    250000
13.0    250000
14.0    250000
15.0    250000
16.0    250000
17.0    250000
18.0    250000
19.0    250000
0.0     250000
Name: faultNumber, dtype: int64
```

In [19]:
```
Y_test.value_counts()
```

Out[19]:
```
20    480000
9     480000
1     480000
2     480000
3     480000
4     480000
5     480000
6     480000
7     480000
8     480000
10    480000
19    480000
11    480000
12    480000
13    480000
14    480000
15    480000
16    480000
17    480000
18    480000
0     480000
Name: faultNumber, dtype: int64
```

In [26]:
```
#lst = [i for i in range(273000000)]
```

In [27]:
```
#import numpy as np
#lst = np.array(lst)
```

In [32]:
```
#lst = lst.reshape(262500,20,52)
#lst[0][1]
```

Out[32]:
```
array([ 52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103])
```

In [20]:
```
train_df = train_df.drop(columns=['faultNumber','simulationRun','sample'])
```

In [21]:
```
test_df = test_df.drop(columns=['faultNumber','simulationRun','sample'])
```

In [23]:
```
#sampling every 20th interval
y_train_lst = []
for i in range(0,Y_train.shape[0],20):
    y_train_lst.append(Y_train.iloc[i])
```

In [25]:
```
y_test_lst = []
for i in range(0,Y_test.shape[0],20):
    y_test_lst.append(Y_test.iloc[i])
```

In [26]:
```
y_train = keras.utils.to_categorical(y_train_lst)
y_test = keras.utils.to_categorical(y_test_lst)
```

In [28]:
```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(train_df)
x_train = scaler.transform(train_df)
x_test = scaler.transform(test_df)
```

In [29]:
```
import numpy as np
x_train = np.array(x_train)
x_test = np.array(x_test)
```

In [31]:
```
#Reshaping the array such that for every 20th interavel the data mathes with y_labe
ls
x_train = x_train.reshape(int(x_train.shape[0]/20),20,52)
x_test = x_test.reshape(int(x_test.shape[0]/20),20,52)
```

In [28]:
```
model = Sequential()
# Configuring the parameters
model.add(LSTM(128,return_sequences=True, input_shape=(20,52)))
# Adding a dropout layer
model.add(Dropout(0.55))

model.add(LSTM(128,return_sequences=True))
model.add(Dropout(0.55))
model.add(LSTM(128,return_sequences=True))
model.add(Dropout(0.55))
model.add(LSTM(128,return_sequences=True))
model.add(Dropout(0.55))
model.add(LSTM(128))

model.add(Dense(21, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_16 (LSTM)               (None, 20, 128)           92672
_________________________________________________________________
dropout_12 (Dropout)         (None, 20, 128)           0
_________________________________________________________________
lstm_17 (LSTM)               (None, 20, 128)           131584
_________________________________________________________________
dropout_13 (Dropout)         (None, 20, 128)           0
_________________________________________________________________
lstm_18 (LSTM)               (None, 20, 128)           131584
_________________________________________________________________
dropout_14 (Dropout)         (None, 20, 128)           0
_________________________________________________________________
lstm_19 (LSTM)               (None, 20, 128)           131584
_________________________________________________________________
dropout_15 (Dropout)         (None, 20, 128)           0
_________________________________________________________________
lstm_20 (LSTM)               (None, 128)               131584
_________________________________________________________________
dense_6 (Dense)              (None, 21)                2709
=================================================================
Total params: 621,717
Trainable params: 621,717
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizer
s.Adam(),metrics=['accuracy'])

history = model.fit(x_train, y_train,batch_size=500,epochs=50,validation_data=(x_te
st, y_test))
```

```
Train on 262500 samples, validate on 504000 samples
Epoch 1/50
262500/262500 [==============================] - 157s 597us/step - loss: 0.9260
- acc: 0.6913 - val_loss: 1.1235 - val_acc: 0.7150
Epoch 2/50
262500/262500 [==============================] - 153s 582us/step - loss: 0.4971
- acc: 0.8252 - val_loss: 1.1124 - val_acc: 0.7339
Epoch 3/50
262500/262500 [==============================] - 151s 575us/step - loss: 0.4299
- acc: 0.8466 - val_loss: 1.0678 - val_acc: 0.7528
Epoch 4/50
262500/262500 [==============================] - 152s 580us/step - loss: 0.3969
- acc: 0.8568 - val_loss: 1.1258 - val_acc: 0.7623
Epoch 5/50
262500/262500 [==============================] - 151s 576us/step - loss: 0.3742
- acc: 0.8649 - val_loss: 1.1463 - val_acc: 0.7676
Epoch 6/50
262500/262500 [==============================] - 153s 583us/step - loss: 0.3607
- acc: 0.8694 - val_loss: 1.1383 - val_acc: 0.7654
Epoch 7/50
262500/262500 [==============================] - 151s 574us/step - loss: 0.3550
- acc: 0.8721 - val_loss: 1.1083 - val_acc: 0.7704
Epoch 8/50
262500/262500 [==============================] - 152s 580us/step - loss: 0.3455
- acc: 0.8753 - val_loss: 1.2066 - val_acc: 0.7714
Epoch 9/50
262500/262500 [==============================] - 151s 575us/step - loss: 0.3415
- acc: 0.8761 - val_loss: 1.1466 - val_acc: 0.7728
Epoch 10/50
262500/262500 [==============================] - 152s 580us/step - loss: 0.3351
- acc: 0.8786 - val_loss: 1.1816 - val_acc: 0.7732
Epoch 11/50
262500/262500 [==============================] - 151s 575us/step - loss: 0.3319
- acc: 0.8800 - val_loss: 1.1894 - val_acc: 0.7742
Epoch 12/50
262500/262500 [==============================] - 153s 581us/step - loss: 0.3288
- acc: 0.8808 - val_loss: 1.2059 - val_acc: 0.7768
Epoch 13/50
262500/262500 [==============================] - 151s 574us/step - loss: 0.3272
- acc: 0.8812 - val_loss: 1.2317 - val_acc: 0.7769
Epoch 14/50
262500/262500 [==============================] - 152s 581us/step - loss: 0.3215
- acc: 0.8838 - val_loss: 1.2981 - val_acc: 0.7785
Epoch 15/50
262500/262500 [==============================] - 151s 574us/step - loss: 0.3199
- acc: 0.8839 - val_loss: 1.2852 - val_acc: 0.7790
Epoch 16/50
262500/262500 [==============================] - 152s 580us/step - loss: 0.3184
- acc: 0.8847 - val_loss: 1.2991 - val_acc: 0.7790
Epoch 17/50
262500/262500 [==============================] - 151s 575us/step - loss: 0.3158
- acc: 0.8849 - val_loss: 1.2532 - val_acc: 0.7795
Epoch 18/50
262500/262500 [==============================] - 153s 581us/step - loss: 0.3148
- acc: 0.8857 - val_loss: 1.2931 - val_acc: 0.7814
Epoch 19/50
262500/262500 [==============================] - 151s 574us/step - loss: 0.3131
- acc: 0.8860 - val_loss: 1.2429 - val_acc: 0.7813
Epoch 20/50
262500/262500 [==============================] - 152s 579us/step - loss: 0.3117
- acc: 0.8868 - val_loss: 1.2884 - val_acc: 0.7810
Epoch 21/50
262500/262500 [==============================] - 151s 576us/step - loss: 0.3075
- acc: 0.8878 - val_loss: 1.2799 - val_acc: 0.7794
```

In [32]:
```
#returns the boolean value True if the predicted point is having the >0.5
pred = model.predict(x_test)
pred = pred > 0.5
```

In [33]:
```
#FDR is fault detection rate which is same as recall
from sklearn.metrics import recall_score
print('FDR:',recall_score(y_test,pred,average='weighted'))
```

```
FDR: 0.7666269841269842
```