In [1]:

```
import random
random.seed(1234)
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import LSTM
import keras.callbacks
from keras import backend as K

def Error(y_pred, y_real):
    y_pred = np.nan_to_num(y_pred, copy = True)
    y_real = np.nan_to_num(y_real, copy = True)
    temp = np.exp(-0.001 * y_real) * np.abs(y_real - y_pred)
    error = np.sum(temp)
    return error

def customLoss(y_pred, y_real):
    return K.sum(K.exp(-0.001 * y_real) * K.abs(y_real - y_pred))
```

```
Using TensorFlow backend.
```

In [5]:

```
# Read in Data
sensor_data = pd.read_csv('/data/phm_data_challenge_2018/train/01_M01_DC_train.csv')
faults_data = pd.read_csv('/data/phm_data_challenge_2018/train/train_faults/01_M01_train_fault_data.csv')
ttf_data = pd.read_csv('/data/phm_data_challenge_2018/train/train_ttf/01_M01_DC_train.csv')

sensor_data = sensor_data.drop(['Tool'], axis = 1)
sensor_data = sensor_data.drop(['Lot'], axis = 1)
```

In [6]:

```
# =============================================================================
# sensor_data = sensor_data.loc[sensor_data.index %10 == 0]
# ttf_data = ttf_data.loc[ttf_data.index %10 == 0]
# =============================================================================
sensor_data.index = range(0,len(sensor_data))
ttf_data.index = range(0,len(ttf_data))

def cutoff(sensor_data, faults_data, ttf_data, column):
    # cut off the tail of the data set that with NaN ttf
    temp = faults_data[faults_data['fault_name'] == column]
    last_failure = temp['time'].values[-1]
    array = np.asarray(sensor_data['time'])
    closest_ind = (np.abs(array - last_failure)).argmin()
    if ((array[closest_ind] - last_failure) != np.abs(array[closest_ind] - last_failure)):
        ind = closest_ind + 1
    elif ((array[closest_ind] - last_failure) == 0):
        ind = closest_ind + 1
    else:
        ind = closest_ind
    sensor_data = sensor_data[:ind]
    ttf_data = ttf_data[:ind]
    faults_data = faults_data[faults_data['fault_name'] == column]
    return sensor_data, ttf_data, faults_data

sensor_fault1, ttf_fault1, faults_fault1 = cutoff(sensor_data, faults_data, ₩
                    ttf_data, 'FlowCool Pressure Dropped Below Limit')

sensor_fault1 = sensor_fault1.fillna(method = 'ffill')
sensor_fault1['recipe'] = sensor_fault1['recipe'] + 200
label = ttf_fault1['TTF_FlowCool Pressure Dropped Below Limit']
```

In [7]:

```
#------------------------------------------------------------------------------
# Capture the trends
temp = ttf_fault1.shift(1)
diff = ttf_fault1['TTF_FlowCool Pressure Dropped Below Limit'] - ₩
        temp['TTF_FlowCool Pressure Dropped Below Limit']
idx = diff[diff > 0].index
trend_start_time = idx.values
trend_start_time = np.insert(trend_start_time, 0, 0)
```

In [ ]:

```
# =============================================================================
# #------------------------------------------------------------------------------
# # One hot encoding
# catagory = ['recipe', 'recipe_step']
# for cat in catagory:
#     one_hot = pd.get_dummies(sensor_fault1[cat])
#     sensor_fault1 = sensor_fault1.drop([cat], axis = 1)
#     sensor_fault1 = sensor_fault1.join(one_hot)
#
# =============================================================================
```

In [8]:

```
# Select data points
def Select(df, y, start_time, num):
    col = []
    y_result = pd.Series()
    for t in range(1, len(start_time)):
        if start_time[t] - start_time[t-1] > num:
           col.append(df[start_time[t] - num: start_time[t]])
           y_result = y_result.append(y[start_time[t] - num: start_time[t]])
        else:
           col.append(df[start_time[t-1]: start_time[t]])
           y_result = y_result.append(y[start_time[t-1]: start_time[t]])
    df_result = pd.concat(col, axis = 0)
    return df_result, y_result

df_select, y_select = Select(sensor_fault1, label, trend_start_time, 2000)
```

In [17]:

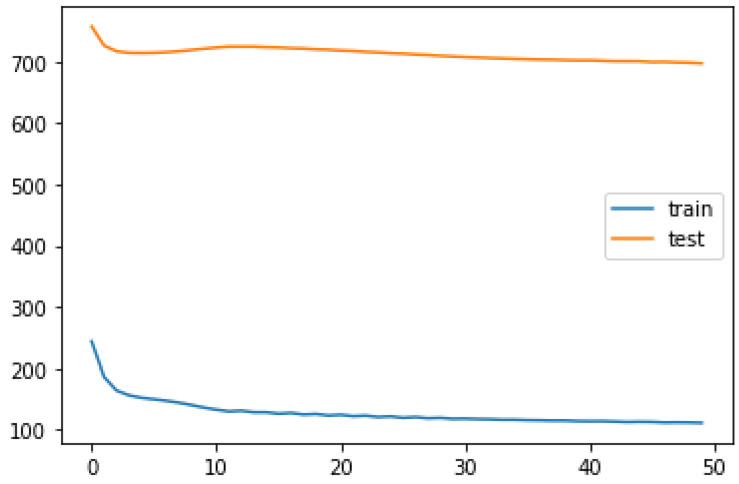
```
# Shift dataset
def series_to_supervised(data, y, n_in=50, dropnan=True):
    data_col = []
    y_col = []
    for i in range (0, n_in):
        data_col.append(data.shift(i))
        y_col.append(y.shift(i))
    result = pd.concat(data_col, axis = 1)
    label = pd.concat(y_col, axis = 1)
    if dropnan:
        result = result[n_in:]
        label = label[n_in:]
    return result, label

df, y = series_to_supervised(df_select, y_select, 10, True)
df_scaler = preprocessing.MinMaxScaler(feature_range = (0,1))
y_scaler = preprocessing.MinMaxScaler(feature_range = (0,1))
feature = df_scaler.fit_transform(df)
label = y_scaler.fit_transform(y)
y_train, y_valid, y_test = label[0:3990], label[16000:], label
X_train, X_valid, y_test = feature[0:3990], feature[16000:], feature
```

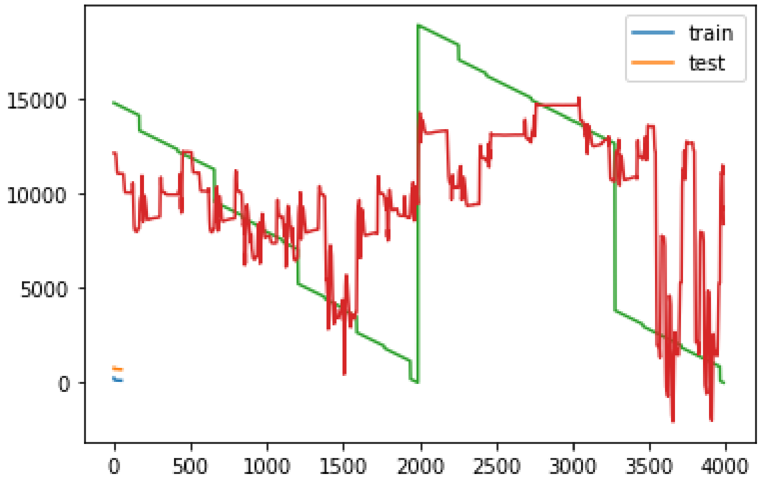
In [23]:

```
#-------------------------------------------------------------------------------
# LSTM
X_train = X_train.reshape((X_train.shape[0], 10, 22))
X_valid = X_valid.reshape((X_valid.shape[0], 10, 22))
model = Sequential()
model.add(LSTM(10, return_sequences=True,  input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(10, return_sequences=True))
model.add(LSTM(10))
model.add(Dense(10))
adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgra
d=False)
model.compile(loss=customLoss, optimizer='adam')
# Early stopping
es = keras.callbacks.EarlyStopping(monitor='val_loss',
                              min_delta=0,
                              patience=2,
                              verbose=0, mode='auto')
history = model.fit(X_train, y_train, epochs=50, batch_size=256, ₩
                    validation_data=(X_valid, y_valid), verbose=2, shuffle=False)

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# scale back the outputs
yhat = model.predict(X_train)
y_pred = y_scaler.inverse_transform(yhat)
y_real = y_scaler.inverse_transform(y_train)
plt.figure()
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.plot(y_real[:,0])
plt.plot(y_pred[:,0])
plt.legend()
plt.show()
# ==============================================================================
# #-------------------------------------------------------------------------------
```

```
Train on 3990 samples, validate on 3990 samples
Epoch 1/50
 - 7s - loss: 243.7516 - val_loss: 757.2459
Epoch 2/50
 - 1s - loss: 186.3028 - val_loss: 726.1496
Epoch 3/50
 - 1s - loss: 164.2367 - val_loss: 716.6260
Epoch 4/50
 - 1s - loss: 156.4315 - val_loss: 714.2699
Epoch 5/50
 - 1s - loss: 152.4263 - val_loss: 713.8829
Epoch 6/50
 - 1s - loss: 149.9738 - val_loss: 714.2871
Epoch 7/50
 - 1s - loss: 147.4757 - val_loss: 715.1885
Epoch 8/50
 - 1s - loss: 144.4435 - val_loss: 716.7355
Epoch 9/50
 - 1s - loss: 140.4854 - val_loss: 718.8893
Epoch 10/50
 - 1s - loss: 136.3310 - val_loss: 720.9022
Epoch 11/50
 - 1s - loss: 133.0018 - val_loss: 722.8392
Epoch 12/50
 - 1s - loss: 130.1185 - val_loss: 724.2918
Epoch 13/50
 - 1s - loss: 131.2237 - val_loss: 724.2649
Epoch 14/50
 - 1s - loss: 128.7606 - val_loss: 724.2845
Epoch 15/50
 - 1s - loss: 128.7253 - val_loss: 723.3703
Epoch 16/50
 - 1s - loss: 126.4038 - val_loss: 722.9046
Epoch 17/50
 - 1s - loss: 127.5929 - val_loss: 721.7719
Epoch 18/50
 - 1s - loss: 125.0286 - val_loss: 721.2496
Epoch 19/50
 - 1s - loss: 125.9368 - val_loss: 719.8744
Epoch 20/50
 - 1s - loss: 123.4876 - val_loss: 719.2807
Epoch 21/50
 - 1s - loss: 124.5827 - val_loss: 717.9682
Epoch 22/50
 - 1s - loss: 122.3281 - val_loss: 717.1480
Epoch 23/50
 - 1s - loss: 123.2211 - val_loss: 715.7884
Epoch 24/50
 - 1s - loss: 120.9588 - val_loss: 714.9169
Epoch 25/50
 - 1s - loss: 121.9992 - val_loss: 713.5747
Epoch 26/50
 - 1s - loss: 119.7004 - val_loss: 712.8213
Epoch 27/50
 - 1s - loss: 120.8766 - val_loss: 711.5164
Epoch 28/50
 - 1s - loss: 118.9082 - val_loss: 710.5875
Epoch 29/50
 - 1s - loss: 119.6316 - val_loss: 709.2737
Epoch 30/50
 - 1s - loss: 118.0274 - val_loss: 708.3032
```

```
Epoch 31/50
 - 1s - loss: 118.2762 - val_loss: 707.3849
Epoch 32/50
 - 1s - loss: 117.5938 - val_loss: 706.4815
Epoch 33/50
 - 1s - loss: 117.2900 - val_loss: 705.6201
Epoch 34/50
 - 1s - loss: 116.5360 - val_loss: 705.0477
Epoch 35/50
 - 1s - loss: 116.5666 - val_loss: 704.3243
Epoch 36/50
 - 1s - loss: 115.8582 - val_loss: 703.6949
Epoch 37/50
 - 1s - loss: 115.6344 - val_loss: 703.2500
Epoch 38/50
 - 1s - loss: 115.1139 - val_loss: 702.9860
Epoch 39/50
 - 1s - loss: 115.1770 - val_loss: 702.3419
Epoch 40/50
 - 1s - loss: 114.3209 - val_loss: 701.8999
Epoch 41/50
 - 1s - loss: 114.0547 - val_loss: 702.0008
Epoch 42/50
 - 1s - loss: 114.2046 - val_loss: 701.2767
Epoch 43/50
 - 1s - loss: 113.5492 - val_loss: 700.5184
Epoch 44/50
 - 1s - loss: 112.8047 - val_loss: 700.4870
Epoch 45/50
 - 1s - loss: 113.1752 - val_loss: 700.3928
Epoch 46/50
 - 1s - loss: 112.9608 - val_loss: 699.0955
Epoch 47/50
 - 1s - loss: 111.9648 - val_loss: 699.3676
Epoch 48/50
 - 1s - loss: 112.2381 - val_loss: 698.6174
Epoch 49/50
 - 1s - loss: 111.9979 - val_loss: 698.0320
Epoch 50/50
 - 1s - loss: 111.4637 - val_loss: 697.0134
```





In [ ]:

```
# # Check correlation between features and labels
def spearman(frame, features):
     spr = pd.DataFrame()
     spr['feature'] = features
     spr['spearman'] = [frame[f].corr(frame['TTF_FlowCool Pressure Dropped Below Limit'], 'spearman') for f in features]
     spr = spr.sort_values('spearman')
     plt.figure(figsize=(6, 0.25*len(features)))
     sns.barplot(data=spr, y='feature', x='spearman', orient='h')
 features = df.columns[0:18]
 spearman(df, features)
 #==============================================================================
```