Design of Soft Sensors Based on Slow Feature Analysis

Capstone Project (CP303)
End-Semester Presentation
May, 2023

Presented By:
Shivam Pandey

Project Mentor:
Dr. V. Jayaram

Problem Statement

- To develop soft-sensors/predictive models based on slow feature analysis.
- Applying the same on different available open industrial datasets.
- To explore the performance of different machine learning algorithms.

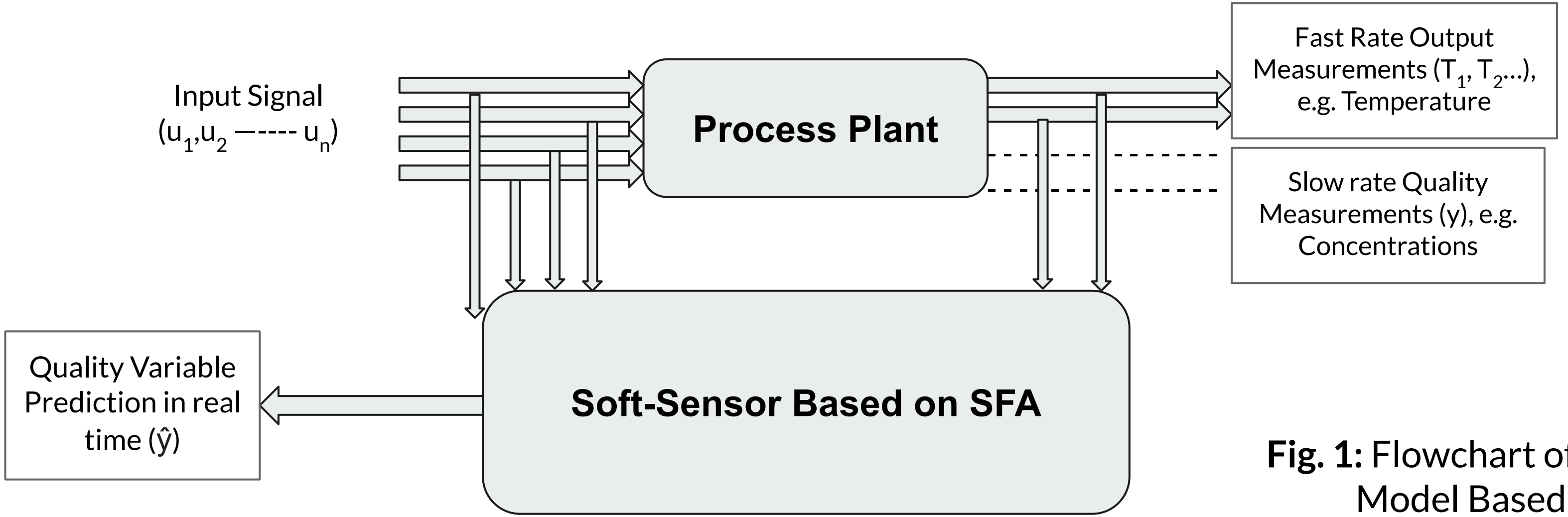


Fig. 1: Flowchart of Soft-Sensor Model Based on SFA

Soft Sensors

- Types: ***Model-based*** (using mathematical models based on first principles - Mechanistic) and ***Data-driven*** (machine learning algorithms analysing historical data)[1].
- Data Driven preferable due to ***high variability and dynamic process, limited knowledge, limited data availability*** and ***ease of implementation.***
- Challenges include ***input selection, data preprocessing, process drift*** (adaptability), ***data collinearity, quantity of data*** and *choice of* ***appropriate models.***
- Various approaches available like ***regression-based models, principal component analysis, slow feature analysis***, ensemble methods like ***random forest***, etc.

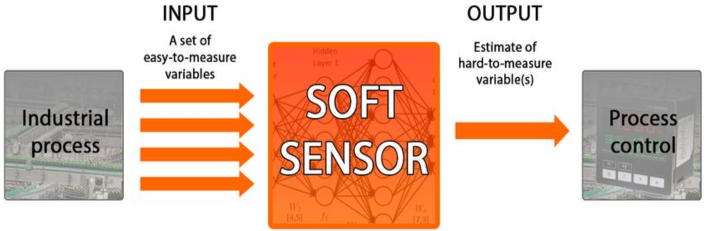


Fig.2 : Working Principle of Soft Sensors (Source)

Generic Approach in Developing Data-Driven Soft Sensors:

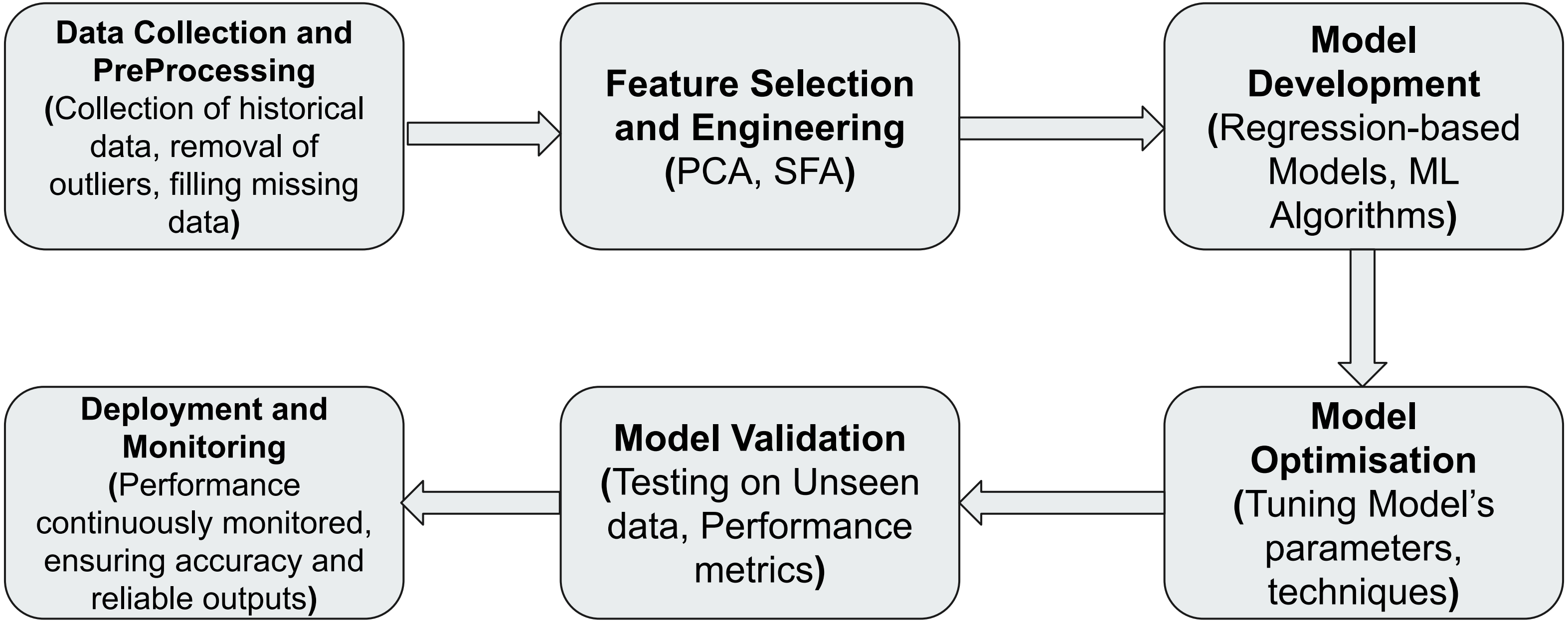
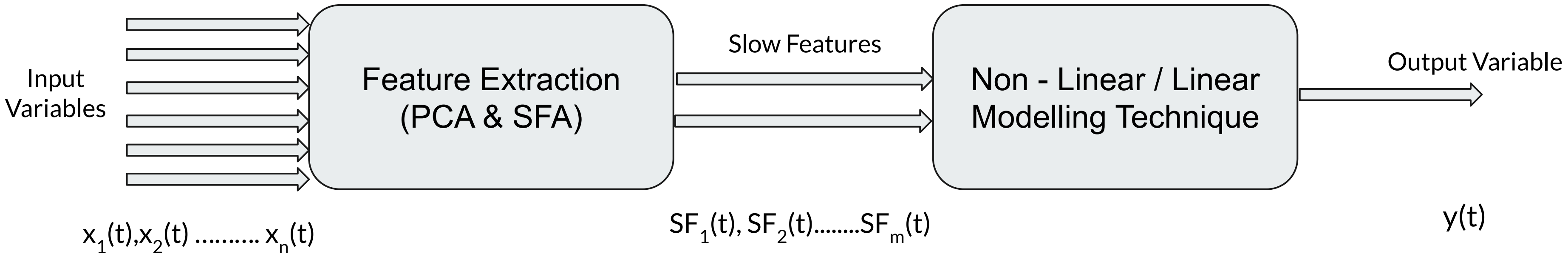


Fig. 3: Approach for Developing Data Driven Soft Sensors

SFA - Slow Feature Analysis

- Noisy data due to *raw material* ***fluctuations***, ***environmental changes, nominal disturbances.***
- Optimal selection of input variables - Improves model performance.
- In addition to ***removing collinearity*** (PCA), extracts ***slowly varying*** variables.
- Captures important underlying trends in process data (***time series***).
- Slowest features - Important Data (Relevant), Fastest Features - ***Noisy*** Data
- ***Dimensional Reduction***, hence less complex.



Example of SFA

Input Signals:

$x_1(t) = -5\sin(t) - 2\cos(5t) + v_1$

$x_2(t) = 3\sin(t) + 0.75\cos(5t) + v_2$

$x_3(t) = -4\sin(t) + \cos(5t) + v_3$

$x_4(t) = 6\sin(t) - \cos(5t) + v_4$

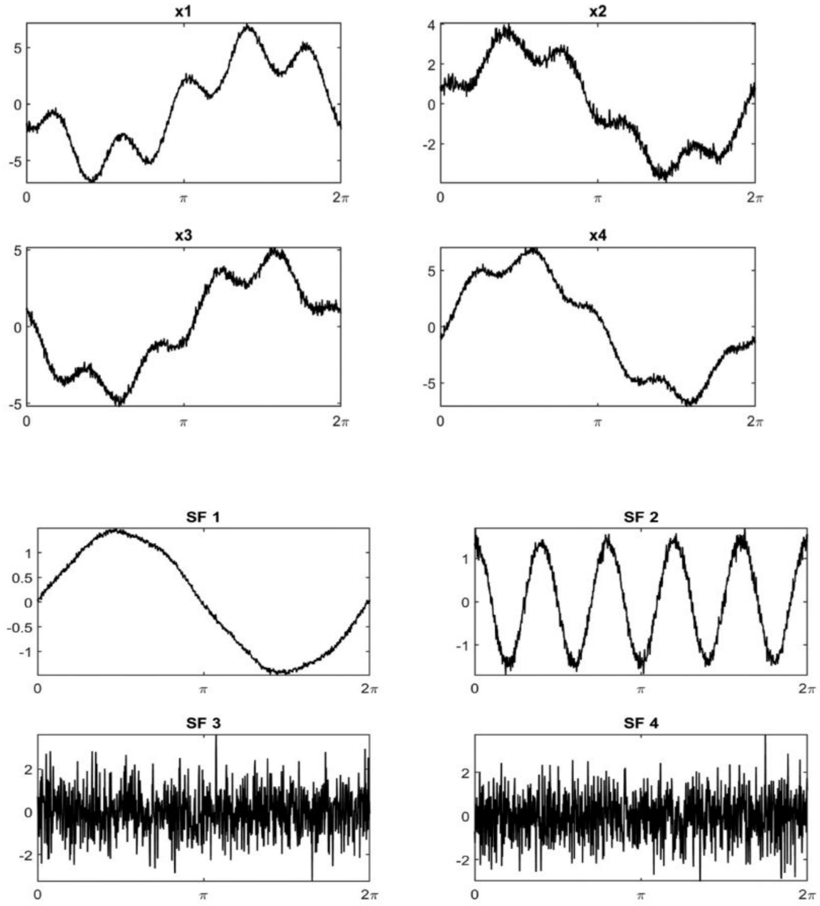


Fig. 4: Input signals and respective extracted slow features

Working Principle of SFA

Given an I dimensional input signal corrupted with noisy and correlated:

$$x(t) = [\, x_1(t)\,,\, x_2(t), \ldots\ldots, x_I(t)\,]$$

Objective: Find an input - output function

$$g(x) = [\, g_1(x), g_2(x), \ldots\ldots g_J(x)\,]$$

To obtain slow features:

$$s(t) = [\, s_1(t), s_2(t), \ldots\ldots s_J(t)\,] \quad \text{where } s_J(t) = g_j(x(t))$$

Aim: Removing Noise

$$\min_{g_j(.)} \Delta(s_j) = \left\langle \dot{s}_j^2 \right\rangle$$

Such that,

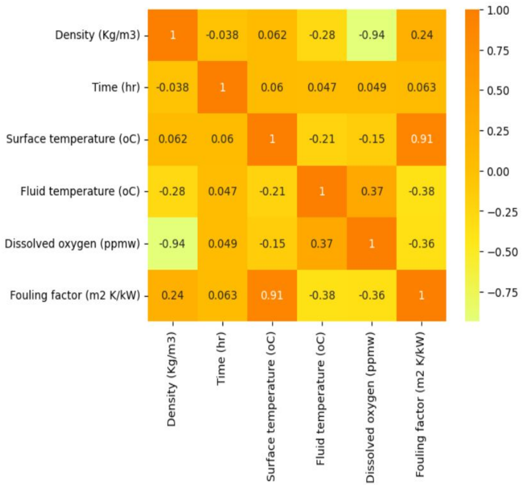
$$\boxed{\begin{array}{l} \langle s_j \rangle = 0 \text{ (zero mean)} \\ \langle s_j^2 \rangle = 1 \text{ (unit variance)} \\ \langle s_{j'} s_j \rangle = 0 \text{ (Decorrelation)} \end{array}}$$

Notation:

$$\boxed{\begin{array}{l} \langle \mathbf{X}(t) \rangle_t = \dfrac{1}{t_1 - t_0} \int_{t_1}^{t_0} \mathbf{X}(t)dt \text{ (for continous)} \\ \approx \dfrac{1}{T} \sum_{t=1}^{T} \mathbf{X}(t) \text{ (for discrete)} \\ \dot{\mathbf{X}}(t) = \dfrac{d\mathbf{X}}{dt} \approx \mathbf{X}(t) - \mathbf{X}(t-1) \end{array}}$$

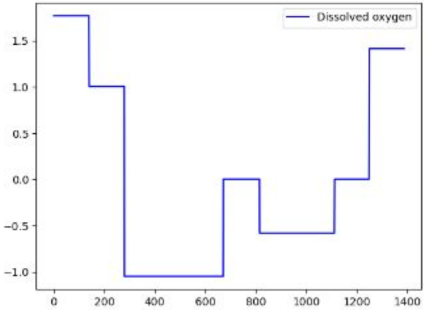
Case Study 1 - Fouling in Heat Exchangers

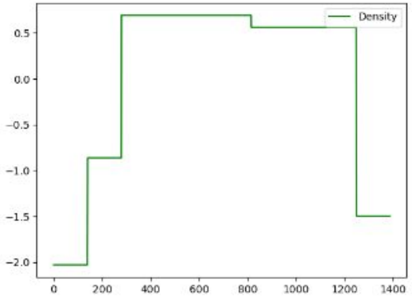
- Accumulation of ***undesired materials*** on the heat transfer surface of the exchanger.
- ***Reduces efficiency***, and leads to decreased performance or failure over time.
- **DataSet Reference:** Asomaning, S., 1990. The role of olefins in fouling of heat exchangers. University of British Columbia, Vancouver [4].
- **Independent Variables** (Features and their Correlation) : Density (0.244), Time (0.308), Surface temperature (0.91), fluid temperature (-0.386), fluid velocity(0), equivalent diameter(0), dissolved oxygen (0.344). [Used Pearson's Correlation] [2].
- **Dependent Variable**: Fouling Factor
- **PCA Results :** Features Include Density, Time, Surface Temperature, Fluid Temperature, Dissolved Oxygen.

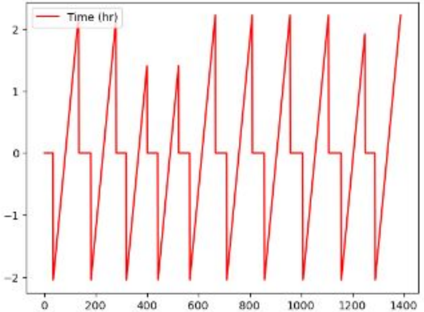


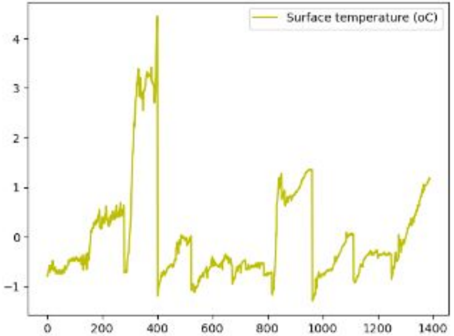
covariance(X,Y) = (sum (X - mean(X)) * (Y - mean(Y))) * 1/(n-1)
Pearson's correlation coefficient: covariance(X,Y) / (stdv(X) * stdv(Y))
stdv - Standard Deviation

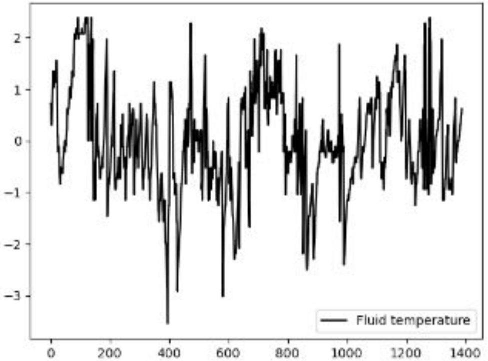
Input Signals

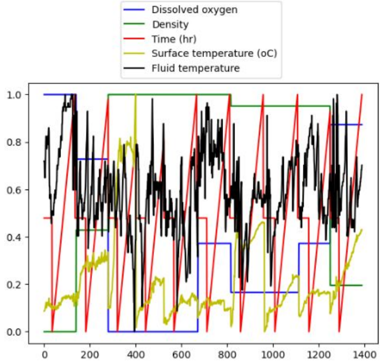




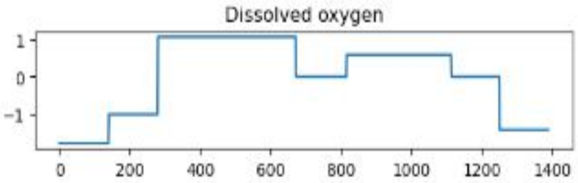


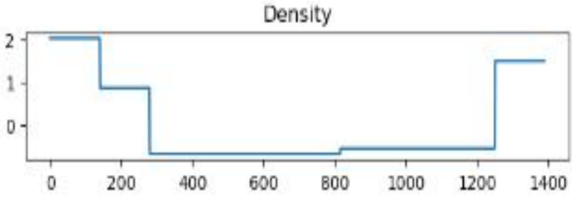


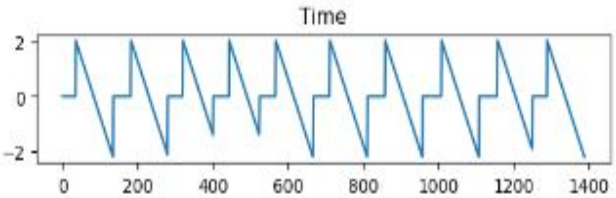


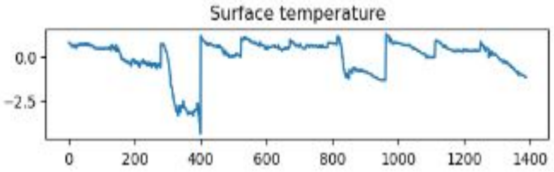


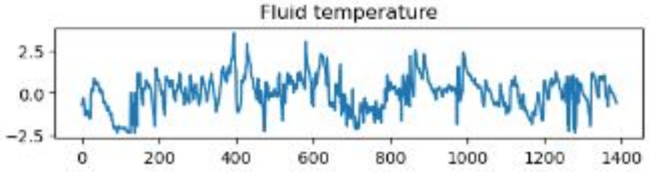
Extracted Slow Feature Signals

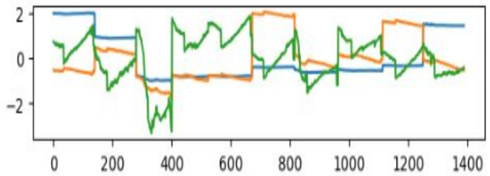












Case Study 2 - Quadruple Tank System

DataSet Reference: Jayaram V, Piyush L, Sachin C, Lorenz T,2017 . Development of moving window state and parameter estimators under maximum likelihood and Bayesian frameworks, Department of Chemical Engineering, Indian Institute of Technology Bombay, India [8].

Independent Variables (Features and their Correlation) : Input Variables (manipulated variables) : Flow1 (0.881), Flow2 (0.249), Flow3 (0.251), Flow4 (0.877). Measured Variables : H1 (0.701), H2 (0.847), H3 (0.309).

Dependent Variable: Measured Variable - Level 4

PCA Results : Features Include Flow1, Flow2, Flow3, Flow4, H1, H2, H3.

Python Library for finding Correlation: from scipy.stats import pearsonr

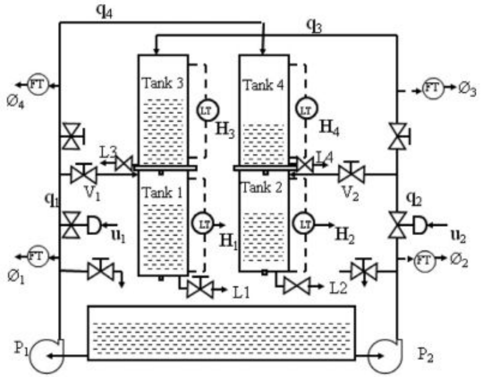
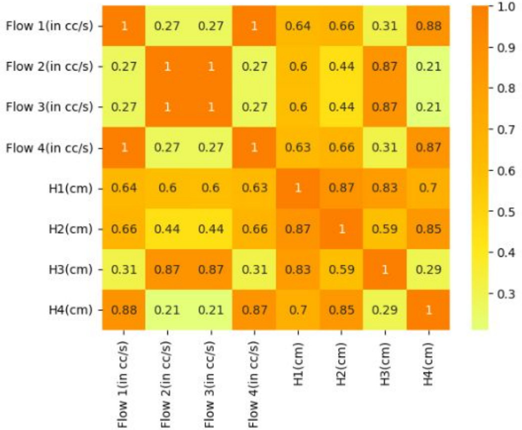
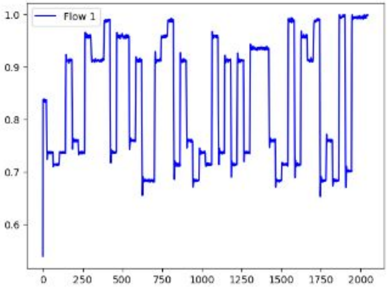
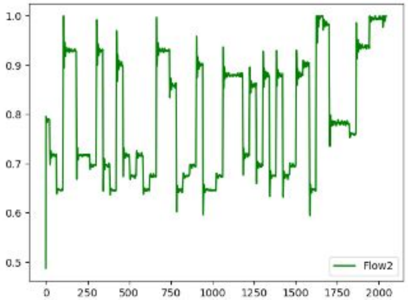


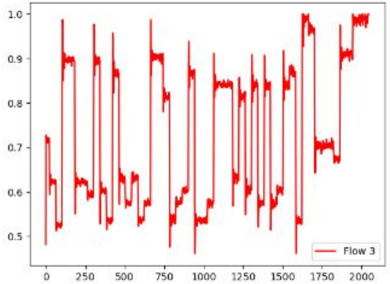
Fig.5 : Quadruple Tank Setup[8]

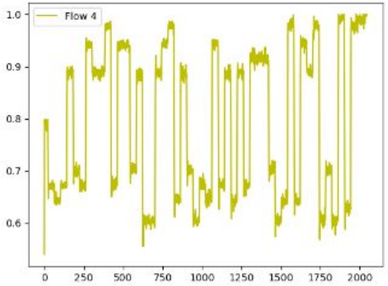


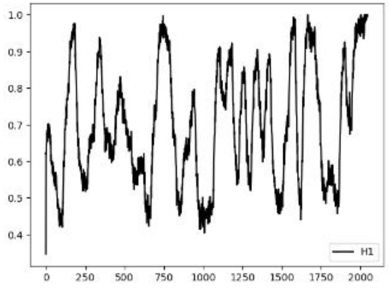
Input Signals

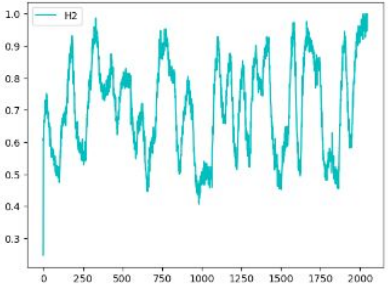


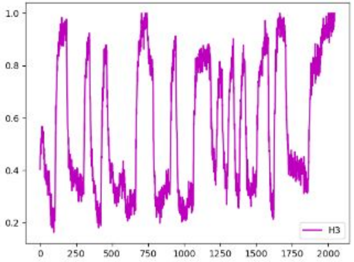


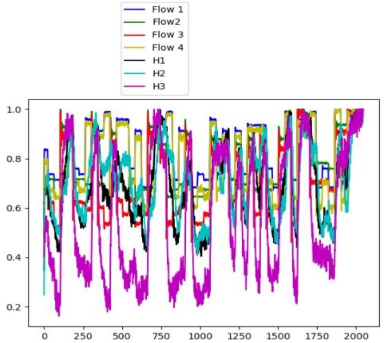




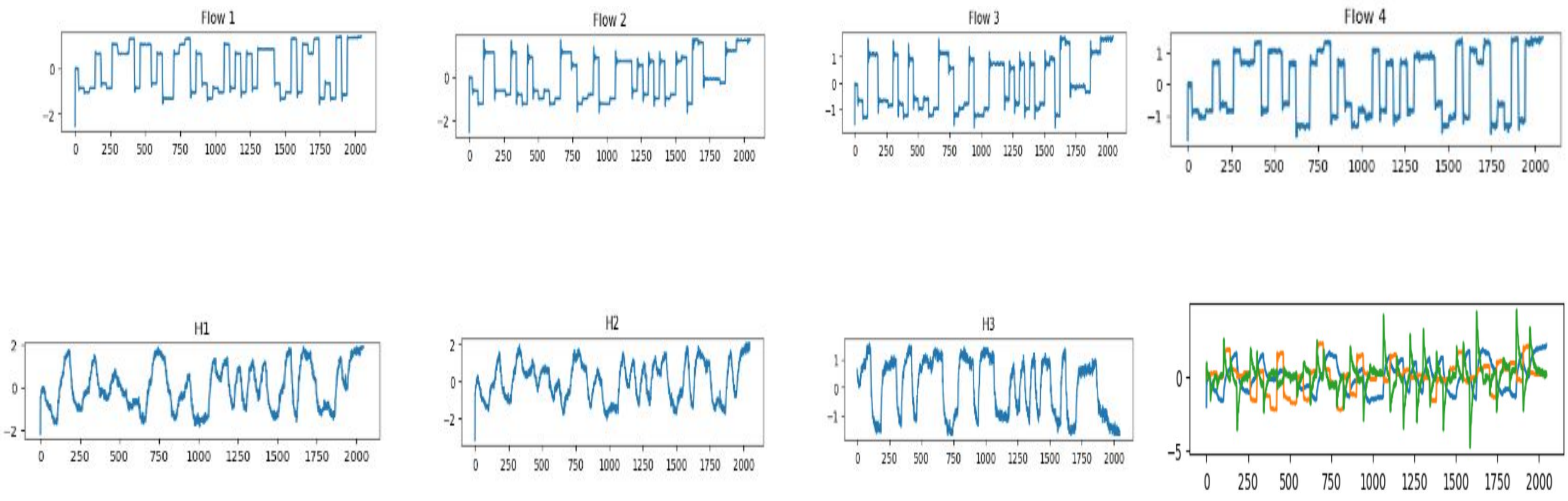








Extracted Slow Feature Signals



Model Development - Linear Regression

- Used mostly when target variable is a real number.
- Works on Principle of Mean Square Error (MSE).

Let $\mathbf{x_i}$ be the independent and $\mathbf{y_i}$ dependent variable.

Hypothesis Function is represented as :

$$h_\theta(x_i) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \ldots\ldots \theta_j x_{ij} \ldots\ldots \theta_n x_{mn}$$

Where θ_j are parameters of hypothesis and x_{ij} (i^{th} training example of j^{th} feature).

Cost Function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Goal : To minimise the cost function.

$$\min_{\theta_0,\theta_1..\theta_n} J(\theta_0, \theta_1 .. \theta_n)$$

$$\frac{\partial J(\theta_j)}{\partial \theta_j} = 0$$

$$\theta = (X^T X)^{-1} X^T y$$

ŷ - predicted value, y - experimental value

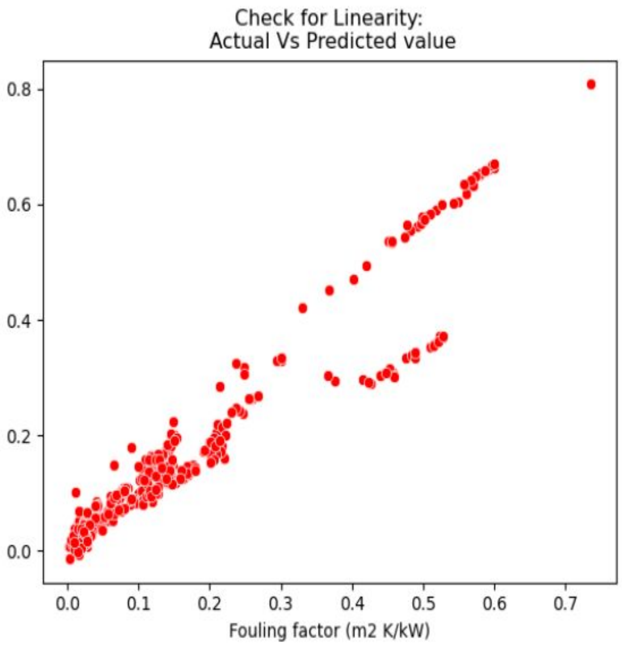


Fig. 6: Plot Between Actual and Predicted Values (CS - 1)

Model Development - Random Forest

- An ***ensemble method***, combines results of multiple decision trees to make predictions.
- Individual decision trees having high variance, combining together in parallel, ***low resultant variance***.
- For classification problem, final output - ***majority voting classifier***.
- For regression problem, final output - ***mean*** of all outputs (Aggregation).
- Greater the number of trees, ***better the accuracy*** and prevents ***overfitting***.
- Decision Tree - consists of root node (starting point), internal nodes (decision based on a feature value), leaf node (final outcome or prediction).

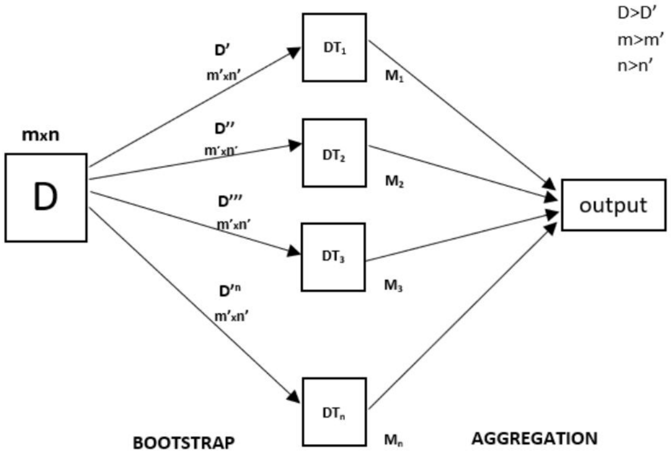


Fig. 7: Random Forest Algorithm
[Source]

Results

Model Validation: Dataset Division: Training Data - 70%, Testing Data - 30%

$R^2 = 1 - A/B$ (Closeness to the fitted regression line)

where $A = \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$ $B = \sum_{i=1}^{m} (y_i - \bar{y}_i)^2$ $\hat{y}$ - predicted value, $\bar{y}$ - mean value

Case Study 1 (Total Features - 5):

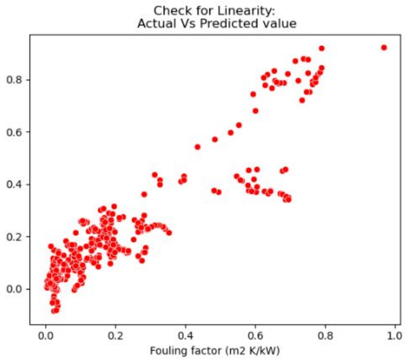
MSE - Mean Square Error

Without SFA, MSE = 0.0022 , R^2 = 0.913

With SFA,

For n = 3, MSE = 0.0079, R^2 = 0.8238

For n = 4, MSE = 0.0041, R^2 = 0.9072



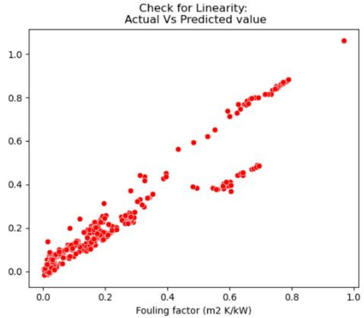


Fig.8: Plot for n = 3 and 4 respectively

Results

Case Study 2 (Total Features = 7):

Without SFA,

Linear Regression, MSE = 0.0031, R^2 = 0.9453.

Random Forest Regression, MSE = 0.0026, R^2 = 0.9551

With SFA,

For n = 3,

Linear Regression, MSE = 0.0065, R^2 = 0.88

Random Forest Regression, MSE = 0.005, R^2 = 0.915

For n = 4,

Linear Regression, MSE = 0.0034, R^2 = 0.94

Random Forest Regression, MSE = 0.0027, R^2 = 0.9535

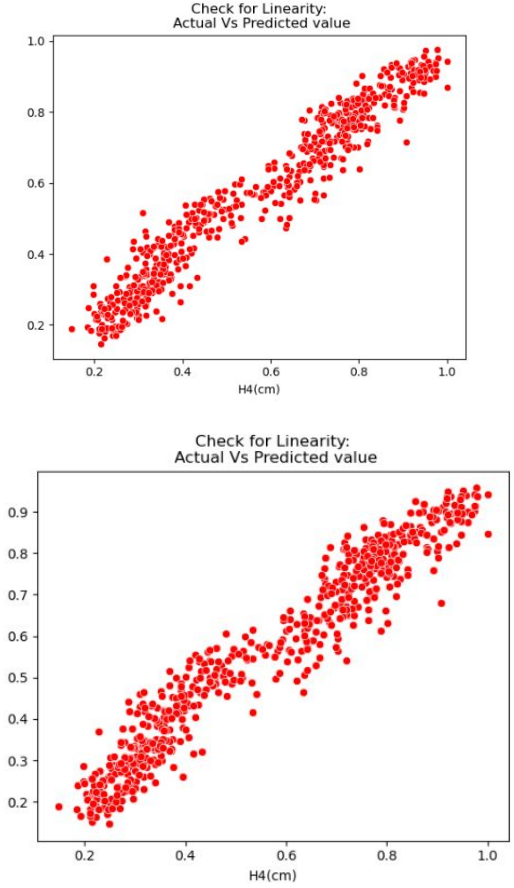


Fig.9: Plot for Without (above) and With SFA (below).

Code Snippets - Case Study 1

```python
import pandas as pd
import pickle as pk
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import preprocessing
import seaborn as sns
from sklearn.linear_model import LinearRegression
from numpy import cov
from scipy.stats import pearsonr
from sksfa import SFA

read_file = pd.read_excel('./Sample_Data_btp1.xlsx')
read_file.to_csv('TrainData.csv',index = None, header = True)
TrainingData = pd.read_csv('./TrainData.csv')
# print(TrainingData)
sum_disso = 0.0
sum_temp = 0.0
sum_stemp = 0.0
sum_t = 0.0
n1 = 0
n2 = 0
n3 = 0
n4 = 0
for i in TrainingData.index:
    if((float)(TrainingData['Dissolved oxygen (ppmw)'][i])!=0.0):
        sum_disso += (float)(TrainingData['Dissolved oxygen (ppmw)'][i])
        n1 += 1
    if(65.0 <= (float)(TrainingData['Fluid temperature (oC)'][i]) <= 95.0):
        sum_temp += (float)(TrainingData['Fluid temperature (oC)'][i])
        n2 += 1
    if(150.0 <= (float)(TrainingData['Surface temperature (oC)'][i]) <= 450.0):
        sum_stemp += (float)(TrainingData['Surface temperature (oC)'][i])
        n3 += 1
    if(15.0 <= (float)(TrainingData['Time (hr)'][i]) <= 48.0):
        sum_t += (float)(TrainingData['Time (hr)'][i])
        n4 += 1
#equivalent diameter is irrelevant
TrainingData = TrainingData.drop(columns = ['Equivalent diameter (m)','Fluid velocity (m/s)'],axis = 1)
avg_dis = sum_disso/n1
avg_temp = sum_temp/n2
avg_stemp = sum_stemp/n3
avg_t = sum_t/n4
for i in TrainingData.index:
    if((float)(TrainingData['Dissolved oxygen (ppmw)'][i])==0.0):
        (TrainingData['Dissolved oxygen (ppmw)'][i]) = avg_dis
    if(not(65.0 <= (float)(TrainingData['Fluid temperature (oC)'][i]) <= 95.0)):
        (TrainingData['Fluid temperature (oC)'][i]) = avg_temp
    if(not(150.0 <= (float)(TrainingData['Surface temperature (oC)'][i]) <= 450.0)):
        (TrainingData['Surface temperature (oC)'][i]) = avg_stemp
    if(not(15.0 <= (float)(TrainingData['Time (hr)'][i]) <= 48.0)):
        (TrainingData['Time (hr)'][i]) = avg_t

X = TrainingData.drop(columns = 'Fouling factor (m2 K/kW)',axis = 1)
Y = TrainingData['Fouling factor (m2 K/kW)']
t = np.linspace(0, 1388, 1389)

#min-max
mean_do = X['Dissolved oxygen (ppmw)'].min()
mean_de = X['Density (Kg/m3)'].min()
mean_t = X['Time (hr)'].min()
mean_st = X['Surface temperature (oC)'].min()
mean_ft = X['Fluid temperature (oC)'].min()
mean_y = Y.min()

std_do = X['Dissolved oxygen (ppmw)'].max()
std_de = X['Density (Kg/m3)'].max()
std_t = X['Time (hr)'].max()
std_st = X['Surface temperature (oC)'].max()
std_ft = X['Fluid temperature (oC)'].max()
std_y = Y.max()
```

```python
for i in X.index:
    X['Dissolved oxygen (ppmw)'][i] = (X['Dissolved oxygen (ppmw)'][i]-mean_do)/(std_do-mean_do)
    X['Density (Kg/m3)'][i] = (X['Density (Kg/m3)'][i]-mean_de)/(std_de-mean_de)
    X['Time (hr)'][i] = (X['Time (hr)'][i]-mean_t)/(std_t-mean_t)
    X['Surface temperature (oC)'][i] = (X['Surface temperature (oC)'][i]-mean_st)/(std_st-mean_st)
    X['Fluid temperature (oC)'][i] = (X['Fluid temperature (oC)'][i]-mean_ft)/(std_ft-mean_ft)
    Y[i] = (Y[i]-mean_y)/(std_y-mean_y)

fig1, ax1 = plt.subplots()
ax1.plot(t,X['Dissolved oxygen (ppmw)'],color='b', label='Dissolved oxygen')
ax1.plot(t,X['Density (Kg/m3)'],color='g', label='Density')
ax1.plot(t,X['Time (hr)'],color='r',label='Time (hr)')
ax1.plot(t,X['Surface temperature (oC)'],color='y',label='Surface temperature (oC)')
ax1.plot(t,X['Fluid temperature (oC)'],color='k',label='Fluid temperature')
ax1.legend(bbox_to_anchor =(0.25, 1.0))

data = np.vstack([X['Dissolved oxygen (ppmw)'],X['Density (Kg/m3)'],X['Time (hr)'],X['Surface temperature (oC)'],X['Fluid temper

# Apply SFA to the data
sfa = SFA(n_components=4)
sfa.fit(data)
slow_features = sfa.transform(data)

# Print the slow features
# print(slow_features)
fig, ax = plt.subplots(3, 1, sharex=True)
fig.subplots_adjust(hspace=0.5)
ax[2].plot(slow_features)
X = slow_features
# print(type(X))
df = pd.DataFrame(X, columns=['A', 'B','C','D'])
X = df
# print(X)
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3, random_state = 23)
X_train_0 = np.c_[np.ones((X_train.shape[0],1)),X_train]
X_test_0 = np.c_[np.ones((X_test.shape[0],1)),X_test]

theta = np.matmul(np.linalg.inv( np.matmul(X_train_0.T,X_train_0) ), np.matmul(X_train_0.T,Y_train))
# The parameters for linear regression model
parameter = ['theta_'+str(i) for i in range(X_train_0.shape[1])]
columns = ['intersect:x_0=1'] + list(X.columns.values)
parameter_df = pd.DataFrame({'Parameter':parameter,'Columns':columns,'theta':theta})
lin_reg = LinearRegression()
lin_reg.fit(X_train,Y_train)
sk_theta = [lin_reg.intercept_]+list(lin_reg.coef_)
parameter_df = parameter_df.join(pd.Series(sk_theta, name='Sklearn_theta'))
y_pred_norm =  np.matmul(X_test_0,theta)
J_mse = np.sum((y_pred_norm - Y_test)**2)/ X_test_0.shape[0]

# R_square
sse = np.sum((y_pred_norm - Y_test)**2)
sst = np.sum((Y_test - Y_test.mean())**2)
R_square = 1 - (sse/sst)
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse)
print('R square obtain for normal equation method is :',R_square)

y_pred_sk = lin_reg.predict(X_test)

#Evaluvation: MSE
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(y_pred_sk, Y_test)

# R_square
R_square_sk = lin_reg.score(X_test,Y_test)
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse_sk)
print('R square obtain for scikit learn library is :',R_square_sk)

f = plt.figure(figsize=(14,5))
ax = f.add_subplot(121)
sns.scatterplot(Y_test,y_pred_sk,ax=ax,color='r')
ax.set_title('Check for Linearity:\n Actual Vs Predicted value')
```

Fig. 10 : Code Snippets - CS1

Code Snippets - Case Study 2

```python
import pandas as pd
import pickle as pk
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import preprocessing
import seaborn as sns
from sklearn.linear_model import LinearRegression
from numpy import cov
from scipy.stats import pearsonr
from sksfa import SFA
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn import svm
from sklearn.model_selection import GridSearchCV



read_file = pd.read_excel('./Data_Quadruple_Tank_System.xlsx')
read_file.to_csv('TrainData1.csv',index = None, header = True)
TrainingData = pd.read_csv('./TrainData1.csv')
TrainingData = TrainingData.drop(columns = ['S.No.','Flow 1','Flow 2','Flow 3','Flow 4','H1','H2','H3','H4'],axis = 1)
# print(TrainingData)
TrainingData['Flow 1(in cc/s)'] = TrainingData['Flow 1(in cc/s)'].fillna(TrainingData['Flow 1(in cc/s)'].mean())
TrainingData['Flow 2(in cc/s)'] = TrainingData['Flow 2(in cc/s)'].fillna(TrainingData['Flow 2(in cc/s)'].mean())
TrainingData['Flow 3(in cc/s)'] = TrainingData['Flow 3(in cc/s)'].fillna(TrainingData['Flow 3(in cc/s)'].mean())
TrainingData['Flow 4(in cc/s)'] = TrainingData['Flow 4(in cc/s)'].fillna(TrainingData['Flow 4(in cc/s)'].mean())
TrainingData['H1(cm)'] = TrainingData['H1(cm)'].fillna(TrainingData['H1(cm)'].mean())
TrainingData['H2(cm)'] = TrainingData['H2(cm)'].fillna(TrainingData['H2(cm)'].mean())
TrainingData['H3(cm)'] = TrainingData['H3(cm)'].fillna(TrainingData['H3(cm)'].mean())
TrainingData['H4(cm)'] = TrainingData['H4(cm)'].fillna(TrainingData['H4(cm)'].mean())

X = TrainingData.drop(columns = 'H4(cm)',axis = 1)
# print(X)
Y = TrainingData['H4(cm)']
# len(X)
# print(Y)
t = np.linspace(0, 2045, 2046)

#NORMALISATION
for i in X.index:
    X['Flow 1(in cc/s)'][i] = (X['Flow 1(in cc/s)'][i]-X['Flow 1(in cc/s)'].min())/(X['Flow 1(in cc/s)'].max()-X['Flow 1(in cc/s
    X['Flow 2(in cc/s)'][i] = (X['Flow 2(in cc/s)'][i]-X['Flow 2(in cc/s)'].min())/(X['Flow 2(in cc/s)'].max()-X['Flow 2(in cc/s
    X['Flow 3(in cc/s)'][i] = (X['Flow 3(in cc/s)'][i]-X['Flow 3(in cc/s)'].min())/(X['Flow 3(in cc/s)'].max()-X['Flow 3(in cc/s
    X['Flow 4(in cc/s)'][i] = (X['Flow 4(in cc/s)'][i]-X['Flow 4(in cc/s)'].min())/(X['Flow 4(in cc/s)'].max()-X['Flow 4(in cc/s
    X['H1(cm)'][i] = (X['H1(cm)'][i]-X['H1(cm)'].min())/(X['H1(cm)'].max()-X['H1(cm)'].min())
    X['H2(cm)'][i] = (X['H2(cm)'][i]-X['H2(cm)'].min())/(X['H2(cm)'].max()-X['H2(cm)'].min())
    X['H3(cm)'][i] = (X['H3(cm)'][i]-X['H3(cm)'].min())/(X['H3(cm)'].max()-X['H3(cm)'].min())
    Y[i] = (Y[i]-Y.min())/(Y.max()-Y.min())
```

```python
fig1, ax1 = plt.subplots()
ax1.plot(t,X['Flow 1(in cc/s)'],color='b', label='Flow 1')
ax1.plot(t,X['Flow 2(in cc/s)'],color='g', label='Flow2')
ax1.plot(t,X['Flow 3(in cc/s)'],color='r',label='Flow 3')
ax1.plot(t,X['Flow 4(in cc/s)'],color='y',label='Flow 4')
ax1.plot(t,X['H1(cm)'],color='k',label='H1')
ax1.plot(t,X['H2(cm)'],color='c',label='H2')
ax1.plot(t,X['H3(cm)'],color='m',label='H3')
ax1.legend(bbox_to_anchor =(0.25, 1.0))
data = np.vstack([X['Flow 1(in cc/s)'],X['Flow 2(in cc/s)'],X['Flow 3(in cc/s)'],X['Flow 4(in cc/s)'],X['H1(cm)'],X['H2(cm)'],X[

# Apply SFA to the data
sfa = SFA(n_components=4)
sfa.fit(data)
slow_features = sfa.transform(data)

# Print the slow features
# print(slow_features)
fig, ax = plt.subplots(3, 1, sharex=True)
fig.subplots_adjust(hspace=0.5)
ax[2].plot(slow_features)
X = slow_features
# print(type(X))
df = pd.DataFrame(X, columns=['A', 'B','C','D'])
X = df
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
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=42)
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

sse = np.sum((y_pred - y_test)**2)
sst = np.sum((y_test - y_test.mean())**2)
R_square = 1 - (sse/sst)
mse = sse/len(y_test)
print('The Mean Square Error(MSE) or J(theta) is: ',mse)
print('R square obtain for normal equation method is :',R_square)
```

Fig. 11 : Code Snippets - CS2

References

1. Lin B., Knudsen J, 2006. A *systematic approach for soft sensor development* [online]. ScienceDirect.[Accessed on 15 January 2023].
2. Ehsan D, Behzad V, 2018. *Applying artificial neural networks for systematic estimation of degree of fouling in heat exchangers*[online]. ScienceDirect. [Accessed on 18 January 2023].
3. Saleh H, Amith K, Muhammad C, 2022. *Novel and robust machine learning approach for estimating the fouling factor in heat exchangers*[online]. ScienceDirect. [Accessed on 21 January 2023].
4. Asomaning, S., 1990. *The role of olefins in fouling of heat exchangers*[online]. University of British Columbia, Vancouver. [Accessed on 28 January 2023].
5. Zhang J, Corrigan J, 2020. *Integrating dynamic slow feature analysis with neural networks for enhancing soft sensor performance*[online]. ScienceDirect. [Accessed on 12 February 2023].
6. C. Shang, B. Huang, F. Yang, D. Huang, 2015. *Probabilistic Slow Feature Analysis based representation learning from massive process data for soft-sensor modelling* [online]. AIChE. [Accessed on 16 March 2023].
7. W. Laurentz, T. J. Sejnowski, 2002. *Slow feature analysis: unsupervised learning of invariances* [online]. Neural computation 14, 715-770. [Accessed on 18 February 2023].
8. Jayaram V, Piyush L, Sachin C, Lorenz T,2017 . *Development of moving window state and parameter estimators under maximum likelihood and Bayesian frameworks* [online]. Department of Chemical Engineering, Indian Institute of Technology Bombay, India. [Accessed on 26 April 2023].

THANK YOU