Fachhochschule Südwestfalen
University of Applied Sciences

Winter Semester (2018/19)

Condition Monitoring of a Hydraulic System using data-driven Machine Learning methods

Course Name: Systems Engineering

Assignment Part: 02

Submitted By:

Sayed Rafay Bin Shah

Matriculation No.: 30043078

Md. Saiful Islam Sajol

Matriculation No.: 30042836

Submitted To:

Prof. Dr.-Ing. Andreas Schwung

Supervised by:

Gavneet Singh Chadha, M.Sc.

Table of Contents

Art.		Contents	Page No.
1.0		Introduction	1
2.0		Collection of Data	1
3.0		Implementing Algorithms	2
	3.1	Importing the datasets and required libraries	2
	3.2	Preprocessing of provided Data	3
4.0		Implementing Deep Neural Network (DNN) Algorithm	4
	4.1	Implementing Neural Network Model for Classification	4
	4.2	Implementing Neural Network Model for Regression	7
5.0		Implementing Support Vector Machine (SVM) Algorithm	10
	5.1	Implementing Support Vector Machine Model for Classification	10
	5.2	Implementing Support Vector Machine Model for Regression	12
6.0		Comparison between results obtained from Neural Network Model and Support Vector Machine Model	18
		References	20

1.0 Introduction:

Fault classification and condition monitoring plays the most important role in predictive maintenance of industrial plants and machineries. The objective of this assignment was condition monitoring and fault classification of a hydraulic system using data-driven machine learning (ML) methods. The ML algorithms used for this purpose are Support Vector Machines (SVM) and Deep Neural Networks (DNN). These two algorithms are developed to perform both classification and regression tasks based on the provided datasets. Afterwards, the results obtained from the different models are compared and analyzed.

2.0 Collection of Data:

The data set was experimentally obtained with a hydraulic test rig. This test rig consists of a primary working and a secondary cooling-filtration circuit, connected via the oil tank [1], [2]. The system cyclically repeats constant load cycles (duration 60 seconds) and measures process values such as pressures, volume flows and temperatures while the condition of four hydraulic components (cooler, valve, pump and accumulator) is quantitatively varied.

The input datasets contains individual data obtained from 17 different sensors. Four types of faults are superimposed with their respective grades of severity, provided in a label dataset named "Profile.csv". All the input and output datasets contain equal number of instances (2205) while the different input sensors possess different attributes per instance. Hence, the input datasets can be termed as Multivariate-Time Series Data. A short illustration of the input dataset characteristics is shown in Table I and the target labels are illustrated in Table II.

Table I Input Data set Characteristics

Sensor	Physical Quantity	Unit	Sampling Rate	No. of Attributes
PS1	Pressure	bar	100 Hz	6000
PS2	Pressure	bar	100 Hz	6000
PS3	Pressure	bar	100 Hz	6000
PS4	Pressure	bar	100 Hz	6000
PS5	Pressure	bar	100 Hz	6000
PS6	Pressure	bar	100 Hz	6000
EPS1	Motor Power	Watt	100 Hz	6000
FS1	Volume Flow	1/min	10 Hz	600
FS2	Volume Flow	1/min	10 Hz	600
TS1	Temperature	°C	1 Hz	60
TS2	Temperature	°C	1 Hz	60
TS3	Temperature	°C	1 Hz	60

Sensor	Physical Quantity	Unit	Sampling Rate	No. of Attributes
TS4	Temperature	°C	1 Hz	60
VS1	Vibration	mm / s	1 Hz	60
CE	Cooling Efficiency	%	1 Hz	60
CP	Cooling Power	kW	1 Hz	60
SE	Efficiency Factor	%	1 Hz	60

Table II Target Label Characteristics

Condition Label	Fault Class	Remark
1. Cooler Condition (%)	3	Close to total failure
	20	Reduced Efficiency
	100	Full Efficiency
2. Valve Condition (%)	100	Optimal switching behavior
	90	Small lag
	80	Severe lag
	73	Close to total failure
3. Internal Pump Leakage	0	No leakage
	1	Weak leakage
	2	Severe leakage
4. Hydraulic Accumulator (bar)	130	Optimal pressure
	115	Slightly reduced pressure
	100	Severely reduced pressure
	90	Close to total failure
5. Stable Flag	0	Conditions were stable
	1	Stable conditions might not have reached yet

3.0 Implementing Algorithms:

The DNN and SVM algorithms were implemented for classification and regression are implemented in Python on Jupyter Notebook. The detailed explanation of implementation are described as follows:

3.1 Importing the datasets and required libraries:

For implementation of DNN, Keras libraries were imported along with Scikit learn libraries on Jupyter notebook. In this operation, Tensorflow was used as backend for Keras. Other relevant

libraries were also imported. The input and label datasets were all provided in .txt format. For convenience, the files were first converted to .csv format and then imported.

3.2 Preprocessing of provided Data:

The data obtained from different sensors were composed of different number of attributes. Hence, to create a more robust input dataset, the mean per instance was computed for each data file, then arranged in a 1D array, and finally labelled as per the sensor name. As a result, the dimension of each sensor data became 2205 x 1 (rows x columns). After this, all the reduced sensor datasets were concatenated column-wise and called under the term X. The target output data was analyzed as a multilabel-multiclass output. These output labels were hence sorted into 5 separate multiclass target labels. Each target label now possesses multiclass continuous target outputs divided into 2205 instances.

3.2.1 Finding Correlation between the Input Sensors data:

To determine the level of correlation between the input sensors data, the correlation matrix using Sklearn is plotted based on Pearson's Correlation Coefficient. The correlation matrix is shown in Fig. 1.

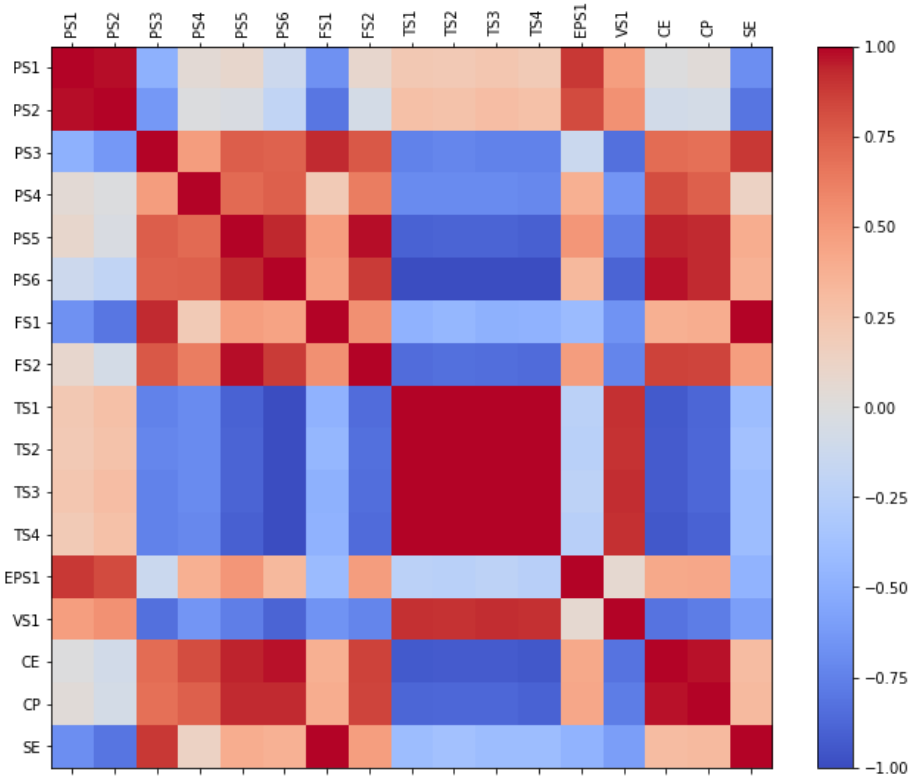

Fig. 1: Correlation matrix between input sensors data sets.

The correlation matrix shows high level of collinearity between certain input sensor data.

4.0 Implementing Deep Neural Network (DNN) Algorithm:

The DNN algorithm is implemented on the Jupyter Notebook platform using Keras libraries with Tensorflow as backend.

4.1 Implementing Neural Network Model for Classification

4.1.1 Converting the Target Labels into Binary Strings

However, the Keras classification metric- categorical_crossentropy is unable to handle multiclass continuous integer target outputs. Hence, each of the target fault label classes are converted from continuous integer values to binary strings of 0 and 1.

Ycc

	Cooler Condition
0	3
1	3
2	3
3	3
...	...
2175	100
2176	100
2177	100

(a)

Y_cooling_cond

	3	20	100
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
...	...	...	...
2175	0	0	1
2176	0	0	1
2177	0	0	1

(b)

Fig. 2: (a) Before and (b) After conversion of target classes into binary value strings.

4.1.2 Train Test Split of Data

The input data sets are split into training and testing samples using the following code with 80% of samples used for training and the remaining 20% for testing.

4.1.3 Feature Scaling:

Feature scaling by means of standardization is performed so that the data has normal distribution properties with zero mean and a standard deviation of one. It reduces the possibility of influence of a high magnitude data on the entire input dataset and also reduces dimensionality of a dataset [3]. This is accomplished using Standard Scaler from SKlearn library.

4.1.4 Designing the NN Model for Classification:

The neural network classifier model is implemented using the Keras library function Sequential(). The model is designed in such a way that the input dataset 'X' is iterated with each of the 5 target labels separately. For this, a loop is generated using the def() function of Sklearn. 'X' denotes the input dataset, 'y' denotes the target label at the output, and 'Label' denotes the kind of fault label the model is iterating against.

4.1.5 Designing the Input, Hidden & Output layers of the Neural Network Model

Input Layer – The input layer consists of 17 nodes followed by the 1st hidden layer.

Hidden Layers – In this task, the neural network model is evaluated for different hidden layer configurations and the model providing the best accuracy is chosen for further analysis. The hidden layers are randomly iterated based on the number of layers (2 or 3) and the number of nodes per layer (24 or 28).

Output Layer – The output layer consists of the number of nodes equal to the number of fault classes of the fault label. Since this model is iterated in loops for five different target labels, the output node is set as equal to the column size of target label 'y'. Hence, for every iteration, the model first detects the number of columns present in the target label and then sets the output nodes as per the number of columns for that target label.

Kernel Initializer – The neural network needs to begin with some initial weights for connections between two consecutive nodes and then iteratively update them to optimal values. 'kernel_initializer' is a term to determine which statistical distribution shall be used for initializing the weights. The library will generate numbers from that statistical distribution and use them as starting weights. In our case, the kernel initializer is set to 'normal'. This means, the normal distribution method will be used to randomly generate initial weights for the neural network [4].

Activation Function – In order to determine whether a neuron should be activated or not and whether the information received by a neuron is relevant or should be ignored, the activation function is used. Without an activation function, the weights of a neuron simply perform linear transformation of input signals and pass it forward to the next node. Hence, the neural network model loses its capacity to perform complex tasks and becomes a linear model. An activation function performs nonlinear transformation at the input of the nodes, making the neurons capable to learn and perform complex tasks [5].
In our model, the activation function used in every layer is 'Softmax'. It is one type of 'Sigmoid' function but possesses the ability of handling more than two classes [5].

4.1.6 Compiling and Fitting the Model

For classification, the model is compiled with the following parameters

Loss Function – This is used to determine the magnitude of error of the model, i.e, how far is the actual result off from the predicted result. Since the target outputs in this task are multiclass outputs and encoded into binary strings, hence categorical_crossentropy is used as the loss function [6].

Optimizer – In neural networks, optimizers play the vital role of optimally updating different model parameters such as, weights and biases in the training process and minimize the error rate of the model. In this task, the Adaptive Moment Estimation (ADAM) is used as the optimizer. This optimizer computes learning rates individually for each model parameter. The 'adam' optimizer obtained from Tensorflow / Keras library is set to default setting (learning rate: 0.001, beta_1: 0.9, beta_2: 0.999, epsilon 1e-08) [7].

Metrics - The classification metric is set to 'categorical_accuracy' to check the accuracy of the model for classifying data based on multiclass target output.

The model is fit using the .fit function in Python. The number of epochs is set to 1000 and batch size is set to 20.

4.1.7 Evaluating Model

The model is evaluated based on the metric 'categorical_accuracy'. Four types of neural network configurations are compared to obtain the most optimum result. The type of configurations used along with their respective accuracies for the different labels is illustrated in Table III.

Table III

	Cooler Condition (% Accuracy)	Valve Condition (% Accuracy)	Internal Pump Leakage (% Accuracy)	Hydraulic Accumulator (% Accuracy)	Stable Flag (% Accuracy)
2 Hidden Layers + 24 Neurons per layer	100.00	85.26	98.41	85.94	92.52
3 Hidden Layers + 24 Neurons per layer	100.00	88.89	98.64	90.70	92.97

	Cooler Condition (% Accuracy)	Valve Condition (% Accuracy)	Internal Pump Leakage (% Accuracy)	Hydraulic Accumulator (% Accuracy)	Stable Flag (% Accuracy)
2 Hidden Layers + 28 Neurons per layer	99.55	86.39	97.28	88.44	93.20
3 Hidden Layers + 28 Neurons per layer	100.00	89.12	98.87	80.95	94.33

Based on the results from Table III, the neural network classifier with **3 Hidden Layers and 24 Neurons** per layers is selected for further analysis and comparison with other machine learning algorithms.

4.2 Implementing Neural Network Model for Regression

4.2.1 Train Test Split of Data

The input data sets are split into training and testing samples with 80% of samples used for training and the remaining 20% for testing.

4.2.2 Feature Scaling:

Feature scaling is performed for regression using MinMax Scaler. This scaling method scales the features within the range of 0 and 1 [8].

4.2.3 Designing the NN Model for Regression:

The neural network regressor model is implemented using the Keras library function Sequential(). Similar to the method used for classifier, this model is also designed in a way that the input dataset 'X' is iterated with each of the 5 target labels separately with a loop is generated using the def() function of Sklearn.

4.2.4 Designing the Input, Hidden & Output layers of the Neural Network Model

Input Layer – The input layer consists of 17 nodes followed by the 1st hidden layer.

Hidden Layers – This model consists of 3 hidden layers with 24 neurons in each layer.

Output Layer – The output consists of a single neuron for regression. The target labels are left as the multiclass integer valued labels for five different fault scenarios.

Kernel Initializer – The kernel initializer is set to 'normal' hence, the normal distribution method will be used to randomly generate initial weights for the neural network.

Activation Function – The Rectified Linear Unit or ReLU is used as the activation function in this model.

4.2.5 Compiling and Fitting the Model

Loss Function & Metric – Mean squared error (mse) is implemented as both the loss function and the regression metric for this task. This will be used to evaluate the model. Mean squared error is the average of the square of errors i.e. the difference between the observed values (y_test) and the predicted ones (y_pred) [9].

Optimizer - The Adaptive Moment Estimation or ADAM optimizer is used for this regression task with its default parameter settings.

The number of epochs is set to 500 and batch size is set to 20.

4.2.6 Evaluating Model

The model is evaluated based on the following parameters:

(a) Mean Squared Error (MSE)

(b) Root Mean Squared Error (RMSE) – RMSE represents how far or how concentrated the regression points are around the line of best fit. It is root measure of the mean squared error [9].

(c) R2 Score – It is called the coefficient of determination for regression. It represents how close the data are located across the fitted regression line [9].

(d) Mean Absolute Error (MAE) – It is a linear score where the error is calculated as an absolute difference between the true values and the predicted values [9].

Table IV illustrated the results obtained from the NN model for the five different target labels

Table IV Neural Network Regressor model scores for target labels

	MSE	RMSE	R2 Score	MAE
Cooler Condition	4.07	2.019	0.99	0.96
Valve Condition	95.28	9.76	0.175	8.573
Pump Leak	0.075	0.274	0.89	0.158
Hydraulic Accumulator	228.53	15.18	0.19	13.33
Stable Flag	0.069	0.263	0.143	0.695

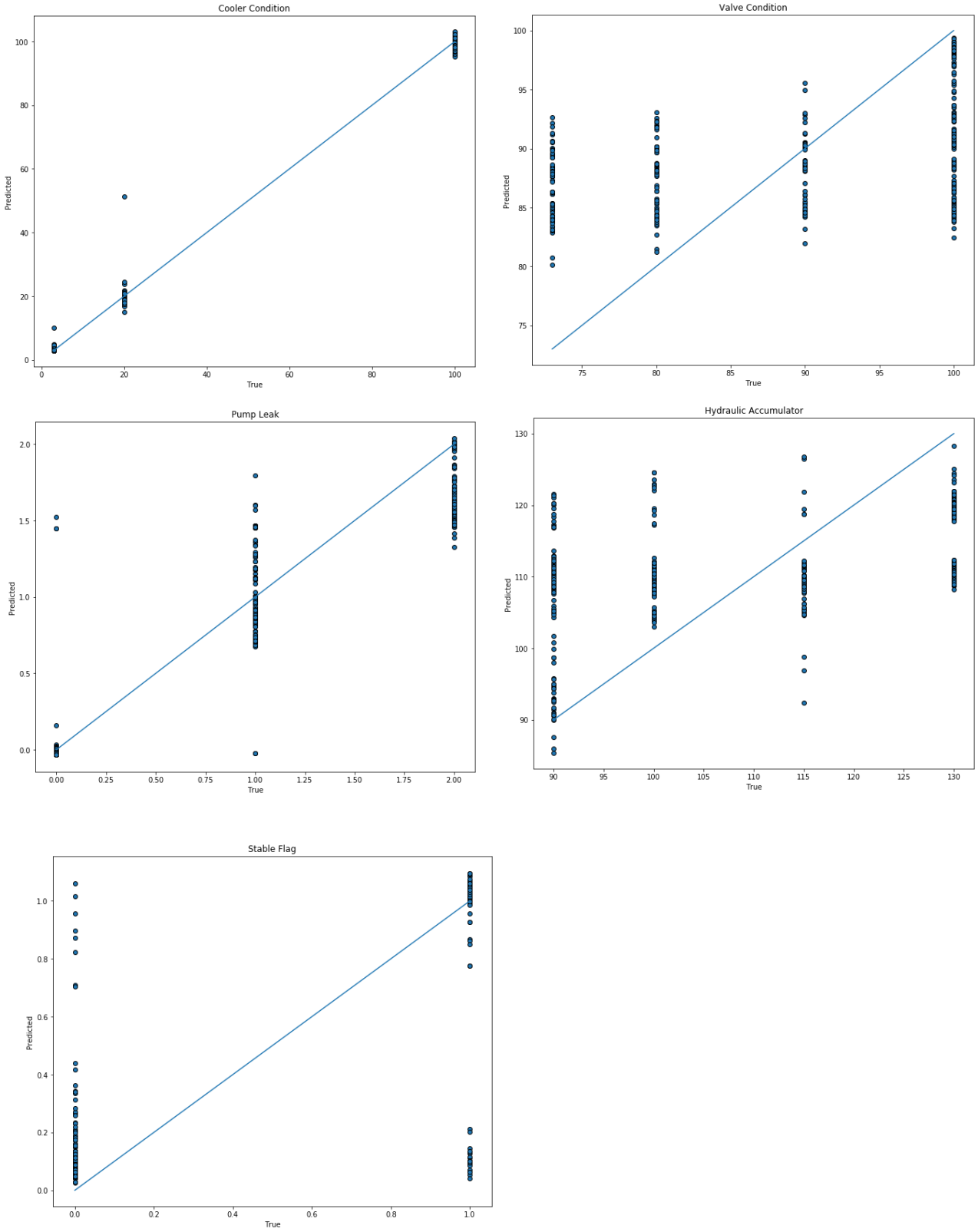

Fig. 3: Line-Scatter plot of True vs Predicted values for five labels: (a) Cooler Condition, (b) Valve Condition, (c) Pump Leak, (d) Hydraulic Accumulator, (e) Stable Flag.

5.0 Implementing Support Vector Machine (SVM) Algorithm:

5.1 Implementing Support Vector Machine Model for Classification

5.1.1 Feature Scaling

Scaling of the datasets is very important before applying the support vector machine algorithm. It helps in avoiding the domination of greater numeric ranges over smaller numbers reducing the errors while performing calculations. Scaling is a pre-processing step of SVM. In this task, the input datasets are scaled using the command StandardScaler from Sci-kit learn [10]. So, both the training and testing input dataset is standardized.

5.1.2 Cross validation

After data mining and data preparation, cross validation is performed. The main purpose of crossvalidation is to distribute the datasets between training data and testing data. The separation of the datasets into training and testing data is important because training and testing of the same data would be a mistake. Training a dataset and then predicting the same dataset would give a perfect score and fail to predict any unseen data, causing overfitting [11]. For this work, 80 percent of the dataset is taken for training and 20 percent is used for testing.

5.1.3 Designing the SVM Model for Classification

Now our dataset is ready for training. A classifier object 'svc' is created for training the 'X_train' and 'X_test' data. From Scikit learn the "*SVC*" function is applied for the training dataset [10]. The accuracy of all the trained dataset is carried out by "*score"* function. It is denoted by 'acculacy'. This accuracy is calculated to compare the training accuracy with the predicted accuracy.

5.1.4 Model Selection

To create a model having a proper classification, we need to tune the parameters of the classifier. This involves tuning the C, kernels, degree, gamma and decision function shape [10].

C: Penalty parameter for considering error for misclassifying dataset [12].

Kernels: This work uses the four kernels (linear, poly,rbf, sigmoid) for calculating the accuracy for each dataset [12].

Degree: This parameter is used for polynomial kernel and ignored for all other kernels [12]

Gamma: This parameter is used for defining the influence of a training example in the classification [12].

Decision function shape: One label versus rest; or one versus one. One-vs-one ('ovo') is always used as multi-class strategy [12].

5.1.5 Faults Prediction

After the dataset is trained, the Classifier algorithm is tested for its accuracy through the prediction of test dataset. Depending upon the training of 'X_train', the classifier will predict the 'faults' for the 'X_test' dataset. The accuracy of the prediction is also carried out by the "score" function *score*.

5.1.6 Model Optimization

Table V shows the accuracy of different classes (i.e. Cooling Failure, Valve Condition, Pump Leaks, Accumulator Condition, and Stable Flag) for four different kernels. All the other parameters (C, degree, gamma, decision function shape) were considered as the default values.

Table V Finding the best kernel
Considering all other parameters at default value (C=default, gamma=default, decision function shape=default)

Accuracy in %	Kernel=linear	rbf	poly	Sigmoid
Cooling Failure	100.00	100.00	100.00	99.55
Valve Condition	67.35	47.85	53.29	47.85
Pump Leaks	97.28	92.97	81.18	63.27
Accumulator Condition	64.85	58.05	58.28	39.91
Stable Flag	86.17	91.16	84.35	56.69

So considering kernel="Linear" is the best model to approach. Now for kernel="linear" we vary the values of C, keeping gamma=default, decision function shape=default)

Table VI Finding the best value of C
Considering Kernel='linear', gamma=default, decision function shape=default

Accuracy in %	C=1	C=10	C=100	C=1000
Cooling Failure	100.00	100.00	100.00	100
Valve Condition	67.35	80.50	93.20	93.42
Pump Leaks	97.28	98.19	98.19	98.41
Accumulator Condition	64.85	71.43	75.51	78.46
Stable Flag	86.17	89.34	90.70	90.93

Table VI shows that accuracy is highest when C=1000 is considered

Now for kernel=”linear”, C=1000, we vary the value of “gamma” where decision function shape=default set

Table VII Finding the optimum value of gamma Considering Kernel=’linear’, C=1000 gamma=default, decision function shape=default				
Accuracy in %	gamma=auto	gamma=0.1	gamma=0.01	gamma=0.001
Cooling Failure	100	100.00	100.00	100.00
Valve Condition	93.42	93.42	93.42	93.42
Pump Leaks	98.41	98.41	98.41	98.41
Accumulator Condition	78.46	78.46	78.46	78.46
Stable Flag	90.93	90.93	90.93	90.93

Above table shows that the values are almost same for different values of gamma. Now we are changing the decision function shape to check for a better result.

Table VIII Considering Kernel=linear, C=1000 & gamma=0.001		
Accuracy in %	ovo	ovr
Cooling Failure	100	100.00
Valve Condition	93.42	93.42
Pump Leaks	98.41	98.41
Accumulator Condition	78.46	78.46
Stable Flag	90.93	90.93

Table VIII shows that, results are almost same for both decision function type.

5.2 Implementing Support Vector Machine Model for Regression

5.2.1 Feature Scaling & Cross validation

These procedures are same as the SVM Classification

5.2.2 Designing the SVM Model for Regression

A classifier object 'svr' is created for training the 'X_train' and 'X_test' data. From Scikit learn the "*SVR*" function is applied for the training dataset. The accuracy of all the trained dataset is carried out by "R2_*score"* function. This accuracy is calculated to compare the true values with the predicted values. Besides two other functions: "mean_squared_error" & "mean_absolute_error" are used to determine the mean squared error and mean absolute error respectively.

5.2.3 Model Selection

To create a model having a proper functioning, we need to tune the parameters of the regression . This involves tuning the C, kernels, degree, epsilon and gamma.

Kernels *(default='rbf')*: It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' is used as default [13].

C *(default=1.0)*: Penalty parameter C of the error term [13].

Degree *(default=3)*: This parameter is used for polynomial kernel and ignored for all other kernels [13].

Gamma *(default='auto')*: This parameter is used for defining the influence of a training example in the regression. It is Kernel coefficient for 'rbf', 'poly' and 'sigmoid' [13].

epsilon *(default=0.1)*:It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value [13].

To determine the best accuracy we have to determine the appropriate values of above parameters.

5.2.4 Finding the values of the parameters:

First, we start our approach considering all other parameters at default except kernel. Below figure shows different values of Mean Square Error (MSE), Mean Absolute Error (MAE) and R2 score for each type of class.

Table IX Finding the best kernel
(Considering C=default(1), epsilion=default(0.1) and Gamma *(default='auto'))*

		Kernel=linear	rbf	poly	sigmoid
Cooling Failure	MSE	5.39	6.08	78.43	175.94
	MAE	1.49	1.26	5.53	9.46
	RMSE	2.32	2.46	8.85	13.26
	R2 Score	0.99	0.99	0.95	0.90
Valve Condition	MSE	72.94	105.92	129.72	154.88
	MAE	6.21	7.84	8.34	9.56
	RMSE	8.54	10.29	11.38	12.44
	R2 Score	0.36	0.08	-0.125	-0.343
Pump Leaks	MSE	0.05	0.07	0.25	53.92
	MAE	0.13	0.15	0.30	5.43
	RMSE	0.23	0.28	0.50	7.34
	R2 Score	0.91	0.88	0.61	-0.80
Accumulator Condition	MSE	194.71	216.44	259.81	274.96
	MAE	10.96	10.98	13.07	13.88
	RMSE	13.95	14.7	16.11	16.58
	R2 Score	0.27	0.19	0.032	-0.024
Stable Flag	MSE	0.10	0.07	0.11	54.48
	MAE	0.254	0.18	0.21	5.40
	RMSE	0.32	0.27	0.33	7.38
	R2 Score	0.48	0.62	0.45	-259.911

Table IX shows that for kernel="linear", overall we get the best result. So we select kernel='linear' for rest of the process.

Now we vary the value of C to get the most suitable value.

Table X Finding the value of C
(Considering Kernel=linear, epsilion=default(0.1) and Gamma *(default='auto'))*

Kernel=linear		C=1	C=10	C=100	C=1000
Cooling Failure	MSE	5.39	4.67	4.82	4.74
	MAE	1.49	1.29	1.24	1.23
	RMSE	2.32	2.16	2.19	2.17
	R2 Score	0.99	0.99	0.99	0.99
Valve Condition	MSE	72.94	22.54	18.63	18.46
	MAE	6.21	3.09	2.36	2.32
	RMSE	8.54	4.74	4.31	4.29
	R2 Score	0.36	0.80	0.83	0.83
Pump Leaks	MSE	0.05	0.05	0.053	0.053
	MAE	0.13	0.12	0.125	0.125
	RMSE	0.23	0.23	0.23	0.23
	R2 Score	0.91	0.91	0.91	0.919
Accumulator Condition	MSE	194.71	143.40	124.32	123.27
	MAE	10.96	9.21	8.74	8.72
	RMSE	13.95	11.97	11.15	11.10
	R2 Score	0.27	0.46	0.53	0.54
Stable Flag	MSE	0.10	0.09	0.095	0.096
	MAE	0.254	0.233	0.232	0.232
	RMSE	0.32	0.30	0.30	0.31
	R2 Score	0.48	0.54	0.54	0.536

Table X shows that better results are found for C=1000 and C=100.Both values of C give similar results. However, we take C=1000 for rest of the process.

Now we find a suitable value of epsilon

Table XI Finding the optimum value of epsilon (Considering kernel="linear", C=1000,gamma=auto)					
Kernel=linear		epsilon=0.1	epsilon=0.2	epsilon=0.3	epsilon=0.5
Cooling Failure	MSE	5.39	4.75	4.75	4.69
	MAE	1.49	1.23	1.23	1.23
	RMSE	2.17	2.18	2.17	2.16
	R2 Score	0.99	0.99	0.99	0.99
Valve Condition	MSE	72.94	18.49	18.48	18.43
	MAE	6.21	2.32	2.321	2.32
	RMSE	4.29	4.3	4.29	4.29
	R2 Score	0.36	0.83	0.83	0.84
Pump Leaks	MSE	0.05	0.054	0.06	0.13
	MAE	0.13	0.133	0.163	0.31
	RMSE	0.231	0.23	0.24	0.36
	R2 Score	0.91	0.91	0.90	0.80
Accumulator Condition	MSE	194.71	123.256	123.40	123.08
	MAE	10.96	8.73	8.73	8.72
	RMSE	11.10	11.10	11.10	11.09
	R2 Score	0.27	0.54	0.54	0.54
Stable Flag	MSE	0.10	0.09	0.09	0.25
	MAE	0.254	0.233	0.24	0.5
	RMSE	0.311	0.306	0.31	0.5
	R2 Score	0.48	0.55	0.53	-0.19

Table XI shows we get the best results for epsilon=0.2. Since we are using linear kernel so we do not need to consider the value of gamma.

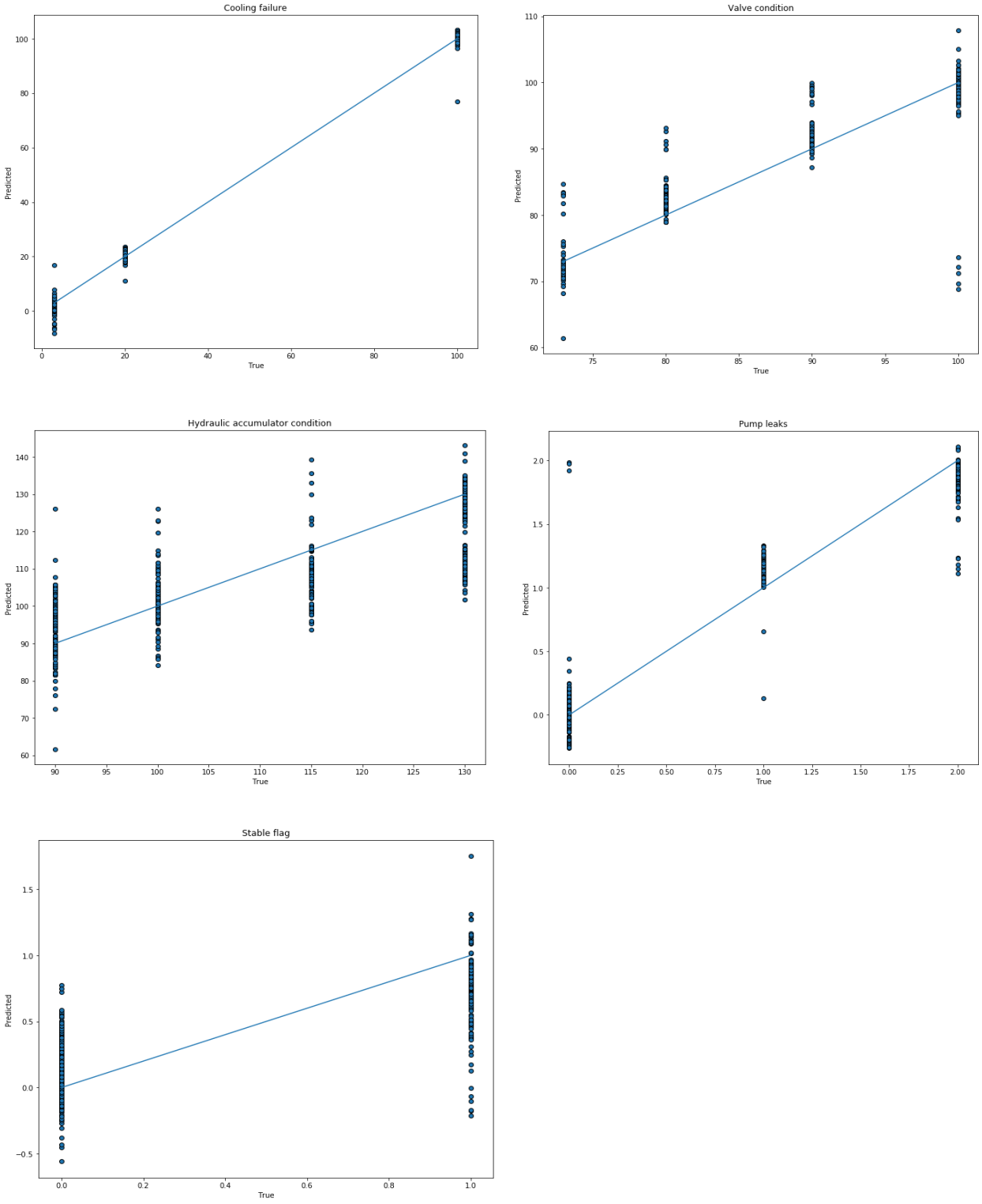

Fig. 4: Line-Scatter plot of True vs Predicted values for five labels: (a) Cooler Condition, (b) Valve Condition, (c) Pump Leak, (d) Hydraulic Accumulator, (e) Stable Flag.

6.0 Comparison between results obtained from Neural Network Model and Support Vector Machine Model

Classification:

Table XII Comparison of results between Neural Network and Support Vector Classifiers

	Neural Network Classifier	Support Vector Classifier
Cooler Condition (%)	100 %	100 %
Valve Condition (%)	88.89 %	93.42 %
Pump Leak (%)	98.64 %	98.41 %
Hydraulic Accumulator (%)	90.70 %	78.46 %
Stable Flag	92.97 %	90.93 %

The model accuracies of Neural Network Classifier and Support Vector Classifier for the given Hydraulic System Dataset is illustrated in Table XII.

In classifying 4 of the 5 target labels, both the models exhibit approximately equal accuracies. However, the deep neural network provides a significantly better accuracy for Hydraulic Accumulator (90.70%) compared to the support vector machine model (78.46%).

Regression:

Table XIII Comparison of results between Neural Network and Support Vector Regressors

		Neural Network Regressor	Support Vector Regressor
Cooler Condition	MSE	4.07	4.75
	RMSE	2.019	2.18
	R2 Score	0.99	0.99
Valve Condition	MSE	95.28	18.49
	RMSE	9.76	4.3
	R2 Score	0.175	0.83
Pump Leak	MSE	0.075	0.054
	RMSE	0.274	0.23
	R2 Score	0.89	0.91
Hydraulic Accumulator	MSE	228.53	123.256
	RMSE	15.18	11.10
	R2 Score	0.19	0.54
Stable Flag	MSE	0.069	0.09
	RMSE	0.263	0.306
	R2 Score	0.143	0.55

The model accuracies of Neural Network Regressor and Support Vector Regressor for the given Hydraulic System Dataset is illustrated in Table XIII.

According to the regression scores provided in Table XIII, the DNN model exhibits slightly better performance than the SVM model in terms of Cooler Condition and Stable Flag. However, the SVM model performs significantly better than the DNN model in terms of the labels Valve Condition and Hydraulic Accumulator. In terms of the label 'Pump Leak', both the model performances are nearly equal.

References

[1] Nikolai Helwig, Eliseo Pignanelli, Andreas Schütze, 'Condition Monitoring of a Complex Hydraulic System Using Multivariate Statistics', in Proc. I2MTC-2015 - 2015 IEEE International Instrumentation and Measurement Technology Conference, paper PPS1-39, Pisa, Italy, May 11-14, 2015, doi: 10.1109/I2MTC.2015.7151267.

[2] N. Helwig, A. Schütze, 'Detecting and compensating sensor faults in a hydraulic condition monitoring system', in Proc. SENSOR 2015 - 17th International Conference on Sensors and Measurement Technology, oral presentation D8.1, Nuremberg, Germany, May 19-21, 2015, doi: 10.5162/sensor2015/D8.1.

[3] S. Asaithambi, "Medium," 4 December 2017. [Online]. Available: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e. [Accessed 26 11 2018].

[4] J. Brownlee, 1 August 2018. [Online]. Available: https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/. [Accessed 27 12 2018].

[5] D. S. Gupta, "Analytics Vidhya," 23 October 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/. [Accessed 19 12 2018].

[6] S. Ronaghan, "Towards Data Science," 18 July 2017. [Online]. Available: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8. [Accessed 23 12 2018].

[7] J. Brownlee, 3 July 2017. [Online]. Available: https:// https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed 27 12 2018].

[8] "Scikit Learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed 2 January 2019].

[9] G. Drakos, "Towards Data Science," 7 August 2018. [Online]. Available: https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0. [Accessed 3 January 2019].

[10] Documentation scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation. [Online] Available: http://scikit-learn.org/stable/documentation.html

[11] Cross-validation: evaluating estimator performance — scikit-learn 0.19.1 documentation. [Online] Available: http://scikit-learn.org/stable/modules/cross_validation.html

[12] Documentation scikit-learn: machine learning in Python [Online]

[13] Documentation scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation [Online]