UNIVERSITY OF ALBERTA

ABHI CHAUHAN

Student ID - 1811065

ADV PROCESS DATA ANALYTICS

CME 660

“Fault detection in Tennessee Eastman Process (TEP) with K-Nearest Neighbor assisted by Neural Network.”

Project Guide: -
Dr. Vinay Prasad
Anjana Puliyanda

Submitted on
12/08/2023

UNIVERSITY OF ALBERTA

Table of Contents

UNIVERSITY OF ALBERTA

List of Figures

List of Tables

1. Introduction

In Chemical industry, the importance of the data (e.g., historical process data) is increasing significantly as it has the power to predict the future of the output in a short time and can prevent the losses in terms of the material and money.

Fault Detection and Diagnosis (FDD) is one of the critical aspects of the chemical Industry as it reduces unprecedented shutdown losses as well as optimizing the losses and cost. The reasons why FDD is important are Advance Process Control (APC), quality control, cost reduction, to increase the operation reliability and minimize human intervention. Statistical process monitoring become widely accepted techniques when it comes to data-driven decision. In that user define the normal operating region by statistical techniques such as PCA or PLS, Then, define sample as faults which doesn't lies within the normal operating region. In recent time, advance techniques such as Neural Network (NN) and Deep learning (DL) become widely accepted in statistical analysis. In this report advance methods are used and discussed in later part.

2. Objective

Main Goal of the project is to find the fault based on the classification techniques taught in class. The pipeline of the project is to first apply PCA (Principal Component Analysis) to extract the features. Then apply Q-statistics and T^2 statistics for further information of the dataset. Then we will use advanced techniques such as Logistic Regression, KNN (K-nearest neighbor) and NN (Nural Network) to detect the fault. However, I came across the techniques where NN assisted by KNN techniques used for FDD has more precision to find each fault with grated accuracy and re-call. This technique involves dividing the fault patterns space into a few sub-spaces using KNN clustering algorithm)[1].

For the validation of the model, test dataset is used for the FDD precision with respect to each fault and accuracy of the same. Results have been discussed in later part of the report. It also involves the assessment of the sampling size of dataset on the result of each techniques used.

3. TEP Dataset overview and Sampling for Analysis.

In this project, TEP (Tennessee Eastman Process)) dataset, a typical open benchmark dataset of chemical industry introduced by Downs and Vogel is used for FDD (Downs, 1993). TEP is a widely used dataset in Process Data Analysis for the FDD and Process Monitoring. The TEP dataset consists of 52 measured variables, which are classified into different categories based on their sampling intervals. The continuous process measurements (variables 1 to 22) have a sampling interval of 3 minutes, while the reactor feed analysis (variables 23 to 28) and purge gas analysis (variables 29 to 36) have a sampling interval of 6 minutes.

[1] [Note: - I proposed fuzzy C-means clustering algorithm instead of KNN, However, due to the computation expensiveness (multiple time code crash and limited RAM issue) KNN was used for the project].

0verall 4 dataset is used in this project. Shape and size are mentioned below table.

Table 1 TEP Dataset Dimensions and overview

	Dataset	Shape	No. of Faults	Simulation	Sample Size
1	Normal Training	250000×55	0	500	500
2	Fault Training	804999×55	20	500	500
3	Normal Testing	480000×55	0	500	960
4	Fault Testing	2787999×55	20	500	960

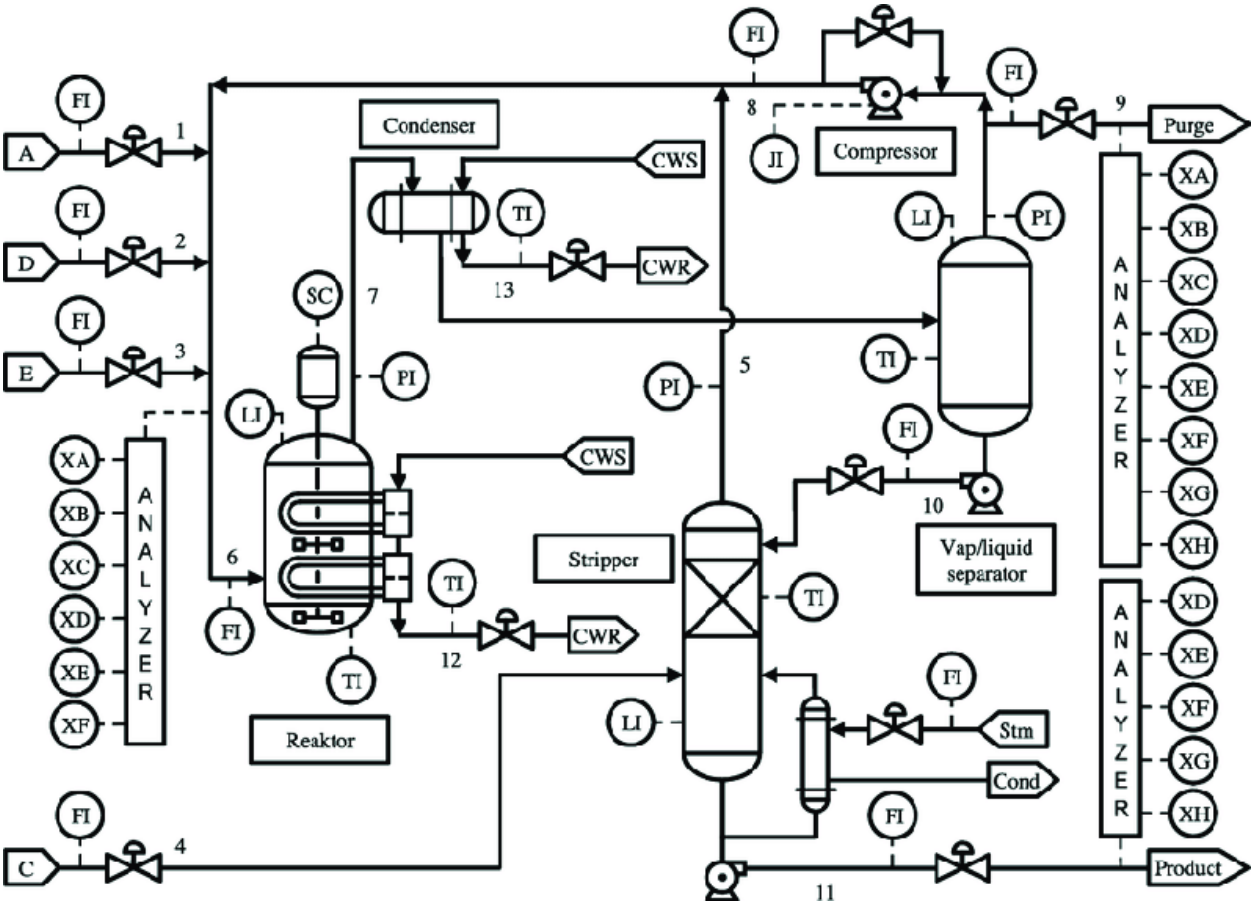

Figure 1TEP process flow diagram

Table 2 Process TAG Description

	Information	Variable	Description	Unit
1	Manipulated Variables (Sampling Frequency = 0.**05** hr)	XMV(1)	D Feed Flow (stream 2) (Corrected Order)	
2		XMV(2)	E Feed Flow (stream 3) (Corrected Order)	
3		XMV(3)	A Feed Flow (stream 1) (Corrected Order)	
4		XMV(4)	A and C Feed Flow (stream 4)	

5		XMV(5)	Compressor Recycle Valve	
6		XMV(6)	Purge Valve (stream 9)	
7		XMV(7)	Separator Pot Liquid Flow (stream 10)	
8		XMV(8)	Stripper Liquid Product Flow (stream 11)	
9		XMV(9)	Stripper Steam Valve	
10		XMV(10)	Reactor Cooling Water Flow	
11		XMV(11)	Condenser Cooling Water Flow	
12		XMV(12)	Agitator Speed	
13	Continuous Process Measurements (Sampling Frequency = 0.**05** hr)	XMEAS(1)	A Feed (stream 1)	kscmh
14		XMEAS(2)	D Feed (stream 2)	kg/hr
15		XMEAS(3)	E Feed (stream 3)	kg/hr
16		XMEAS(4)	A and C Feed (stream 4)	kscmh
17		XMEAS(5)	Recycle Flow (stream 8)	kscmh
18		XMEAS(6)	Reactor Feed Rate (stream 6)	kscmh
19		XMEAS(7)	Reactor Pressure	kPa gauge
20		XMEAS(8)	Reactor Level	%
21		XMEAS(9)	Reactor Temperature	Deg C
22		XMEAS(10)	Purge Rate (stream 9)	kscmh
23		XMEAS(11)	Product Sep Temp	Deg C
24		XMEAS(12)	Product Sep Level	%
25		XMEAS(13)	Prod Sep Pressure	kPa gauge
26		XMEAS(14)	Prod Sep Underflow (stream 10)	m3/hr
27		XMEAS(15)	Stripper Level	%
28		XMEAS(16)	Stripper Pressure	kPa gauge
29		XMEAS(17)	Stripper Underflow (stream 11)	m3/hr
30		XMEAS(18)	Stripper Temperature	Deg C
31		XMEAS(19)	Stripper Steam Flow	kg/hr
32		XMEAS(20)	Compressor Work	kW
33		XMEAS(21)	Reactor Cooling Water Outlet Temp	Deg C
34		XMEAS(22)	Separator Cooling Water Outlet Temp	Deg C
35	Sampled Process Measurements (Sampling Frequency = 0.1 hr)	XMEAS(23)	Component A	
36		XMEAS(24)	Component B	
37		XMEAS(25)	Component C	
38		XMEAS(26)	Component D	
39		XMEAS(27)	Component E	
40		XMEAS(28)	Component F	
41	Purge Gas Analysis (Stream 9) Sampling Frequency = 0.1 hr	XMEAS(29)	Component A	
42		XMEAS(30)	Component B	
43		XMEAS(31)	Component C	
44		XMEAS(32)	Component D	
45		XMEAS(33)	Component E	
46		XMEAS(34)	Component F	
47		XMEAS(35)	Component G	

48		XMEAS(36)	Component H	
49	Product Analysis (Stream 11) Sampling Frequency = 0.25 hr	XMEAS(37)	Component D	
50		XMEAS(38)	Component E	
51		XMEAS(39)	Component F	
52		XMEAS(40)	Component G	
53		XMEAS(41)	Component H	
	Process Disturbances			
1		IDV(1)	A/C Feed Ratio, B Composition Constant (Stream 4) Step	
2		IDV(2)	B Composition, A/C Ratio Constant (Stream 4) Step	
3		IDV(3)	D Feed Temperature (Stream 2) Step	
4		IDV(4)	Reactor Cooling Water Inlet Temperature Step	
5		IDV(5)	Condenser Cooling Water Inlet Temperature Step	
6		IDV(6)	A Feed Loss (Stream 1) Step	
7		IDV(7)	C Header Pressure Loss - Reduced Availability (Stream 4) Step	
8		IDV(8)	A, B, C Feed Composition (Stream 4) Random Variation	
9		IDV(9)	D Feed Temperature (Stream 2) Random Variation	
10		IDV(10)	C Feed Temperature (Stream 4) Random Variation	
11		IDV(11)	Reactor Cooling Water Inlet Temperature Random Variation	
12		IDV(12)	Condenser Cooling Water Inlet Temperature Random Variation	
13		IDV(13)	Reaction Kinetics Slow Drift	
14		IDV(14)	Reactor Cooling Water Valve Sticking	
15		IDV(15)	Condenser Cooling Water Valve Sticking	
16		IDV(16)	Unknown	
17		IDV(17)	Unknown	
18		IDV(18)	Unknown	
19		IDV(19)	Unknown	
20		IDV(20)	Unknown	

3.1 Sample Selection

In this work, Fault number 3, 9 and 15 are not considered as they are difficult to detect due to almost no change in the statistical data point of view. (Zhang, 2009,)

For this training dataset, we only used first 50 simulationsRun from the 500 simulationRun. It has been observed that while handling the entire dataset, Kaggle Notebook can perform the PCA and Logistic Regression like seldom computation expensiveness. However, when we use classifier like KNN, NN and KNN-NN it fails to do so due to limited amount of GPU/RAM and Data processing restriction. To improve the overall accuracy of the dataset, sample size further cropped from

sample 20 to 500 for training dataset as the actual faults are start after sample 20th for the training dataset. (Seongmin Heo, 2019)

For the testing dataset, we only used 10 simulationRun to check accuracy and precision of each fault. Again, we reduced the data by taking sample from 160 to 960 for each simulationRun as actual fault start from 120th sample. (Seongmin Heo, 2019)

Table 3 Downsize Sample proposed for the project

	Normal Training	**Fault Training**	**Normal Test**	**Fault test**
SimulationRun	50	50 per Fault	10	10 per fault
Sample	500	480 (20 to 500)	10	840(120 to 960)

4. Methodology

4.1 Data Pre-Processing:

Missing Data: Before start the work, data must be check whether there is missing data or not. By using *isnull().sum()* function on the dataset we can get the missing data value. For the all the dataset there is no missing values.

Table 4 Missing Data or null values in Dataset

Normal Training	**Fault Training**	**Normal Testing**	**Fault Testing**
Missing Data:	Missing Data:	Missing Data:	Missing Data:
faultNumber 0	faultNumber 0	faultNumber 0	faultNumber 0
simulationRun 0	simulationRun 0	simulationRun 0	simulationRun 0
sample 0	sample 0	sample 0	sample 0
xmeas_1 0	xmeas_1 0	xmeas_1 0	xmeas_1 0
xmeas_2 0	xmeas_2 0	xmeas_2 0	xmeas_2 0
xmeas_3 0	xmeas_3 0	xmeas_3 0	xmeas_3 0
xmeas_4 0	xmeas_4 0	xmeas_4 0	xmeas_4 0
xmeas_5 0	xmeas_5 0	xmeas_5 0	xmeas_5 0
xmeas_6 0	xmeas_6 0	xmeas_6 0	xmeas_6 0
xmeas_7 0	xmeas_7 0	xmeas_7 0	xmeas_7 0
xmeas_8 0	xmeas_8 0	xmeas_8 0	xmeas_8 0
xmeas_9 0	xmeas_9 0	xmeas_9 0	xmeas_9 0
xmeas_10 0	xmeas_10 0	xmeas_10 0	xmeas_10 0
xmeas_11 0	xmeas_11 0	xmeas_11 0	xmeas_11 0
xmeas_12 0	xmeas_12 0	xmeas_12 0	xmeas_12 0
xmeas_13 0	xmeas_13 0	xmeas_13 0	xmeas_13 0
xmeas_14 0	xmeas_14 0	xmeas_14 0	xmeas_14 0
xmeas_15 0	xmeas_15 0	xmeas_15 0	xmeas_15 0
xmeas_16 0	xmeas_16 0	xmeas_16 0	xmeas_16 0
xmeas_17 0	xmeas_17 0	xmeas_17 0	xmeas_17 0
xmeas_18 0	xmeas_18 0	xmeas_18 0	xmeas_18 0
	xmeas_19 0	xmeas_19 0	xmeas_19 0

xmeas_19	0	xmeas_20	0	xmeas_20	0	xmeas_20	0
xmeas_20	0	xmeas_21	0	xmeas_21	0	xmeas_21	0
xmeas_21	0	xmeas_22	0	xmeas_22	0	xmeas_22	0
xmeas_22	0	xmeas_23	0	xmeas_23	0	xmeas_23	0
xmeas_23	0	xmeas_24	0	xmeas_24	0	xmeas_24	0
xmeas_24	0	xmeas_25	0	xmeas_25	0	xmeas_25	0
xmeas_25	0	xmeas_26	0	xmeas_26	0	xmeas_26	0
xmeas_26	0	xmeas_27	0	xmeas_27	0	xmeas_27	0
xmeas_27	0	xmeas_28	0	xmeas_28	0	xmeas_28	0
xmeas_28	0	xmeas_29	0	xmeas_29	0	xmeas_29	0
xmeas_29	0	xmeas_30	0	xmeas_30	0	xmeas_30	0
xmeas_30	0	xmeas_31	0	xmeas_31	0	xmeas_31	0
xmeas_31	0	xmeas_32	0	xmeas_32	0	xmeas_32	0
xmeas_32	0	xmeas_33	0	xmeas_33	0	xmeas_33	0
xmeas_33	0	xmeas_34	0	xmeas_34	0	xmeas_34	0
xmeas_34	0	xmeas_35	0	xmeas_35	0	xmeas_35	0
xmeas_35	0	xmeas_36	0	xmeas_36	0	xmeas_36	0
xmeas_36	0	xmeas_37	0	xmeas_37	0	xmeas_37	0
xmeas_37	0	xmeas_38	0	xmeas_38	0	xmeas_38	0
xmeas_38	0	xmeas_39	0	xmeas_39	0	xmeas_39	0
xmeas_39	0	xmeas_40	0	xmeas_40	0	xmeas_40	0
xmeas_40	0	xmeas_41	0	xmeas_41	0	xmeas_41	0
xmeas_41	0	xmv_1	0	xmv_1	0	xmv_1	0
xmv_1	0	xmv_2	0	xmv_2	0	xmv_2	0
xmv_2	0	xmv_3	0	xmv_3	0	xmv_3	0
xmv_3	0	xmv_4	0	xmv_4	0	xmv_4	0
xmv_4	0	xmv_5	0	xmv_5	0	xmv_5	0
xmv_5	0	xmv_6	0	xmv_6	0	xmv_6	0
xmv_6	0	xmv_7	0	xmv_7	0	xmv_7	0
xmv_7	0	xmv_8	0	xmv_8	0	xmv_8	0
xmv_8	0	xmv_9	0	xmv_9	0	xmv_9	0
xmv_9	0	xmv_10	0	xmv_10	0	xmv_10	0
xmv_10	0	xmv_11	0	xmv_11	0	xmv_11	0
xmv_11	0						

Downsizing of the sample: Downsize of the train and test dataset was done by following manner.

1) Combine the Normal and Fault train data first and make new data frame as ‘NTFTN_500’ by (.concat) function
2) Extract the first 50 simulationRun out of the 500 simulationRun.
3) Remove the all the fault rows contain fault 3,9 and 15 by applying the operator (!=) on the faultNumber
4) Select the sample from the 20 to 500 from the 0 to 500 sample columns by using numpy dictionary.
5) Assign the variable name NTFT500_train_features to all the features of the dataset by the (.iloc) function. (column number 3 to 55)

6) Assign the variable name NTFT500_train_lable to all the features of the dataset by the (.iloc) function. (column number 0)
7) Combine the Normal and Fault test data and make new data frame as NTFTS50 by (.concat) function
8)
9) Extract the first 10 simulationRun out of the 500 simulationRun.
10) Remove the all the fault rows contain fault 3,9 and 15 by applying the operator (!=) on the faultNumber
11) Select the sample from the 160 to 960 from the 0 to 500 sample columns by using numpy dictionary.
12) Assign the variable name NTFT50_test_features to all the features of the dataset by the (.iloc) function. (column number 3 to 55)
13) Assign the variable name NTFT50_test_lable to all the features of the dataset by the (.iloc) function. (column number 0)

Final Shape of the Datasets:

Training Dataset = 18 faults (0 represent normal data) * 50 simulationRun * 481 sample =
= 432900 × 55
Test Dataset = 18 faults (0 represent normal data) * 10 simulationRun * 801 sample =
= 172800 × 55

4.2 PCA (Principal Component Analysis):

Primary use of PCA is to dimensionality reduction while retaining the maximum information. It also helps to reduce instability in the regression as highly correlated data create the instability in the model, which we will use in this project as Linear Regression.

1) INPUT dataset dimensions: - 432900 × 52
2) OUTPUT dimensions: - 432900 × 25
3) Loss Function: - As PCA is the unsupervised techniques there is no loss function. However, we can count the loss of the data via Reconstruction Error (RSME) which found as 0.0434. It seems good for almost half dimension reduction of dataset.
4) Parameters: -For the PCA number of the parameters is same as number of the components i.e., 25

Result of the PCA from the code:

Table 5 Explained variance and Cumulative Explained variance of PCA component

Explained Variance for PCA1: 0.2505	Cumulative Explained Variance up to PCA1: 0.2505
Explained Variance for PCA2: 0.1943	Cumulative Explained Variance up to PCA2: 0.4448
Explained Variance for PCA3: 0.0766	Cumulative Explained Variance up to PCA3: 0.5214
Explained Variance for PCA4: 0.0572	Cumulative Explained Variance up to PCA4: 0.5786
Explained Variance for PCA5: 0.0387	Cumulative Explained Variance up to PCA5: 0.6173

Explained Variance for PCA6: 0.0365	Cumulative Explained Variance up to PCA6: 0.6538
Explained Variance for PCA7: 0.0270	Cumulative Explained Variance up to PCA7: 0.6808
Explained Variance for PCA8: 0.0254	Cumulative Explained Variance up to PCA8: 0.7062
Explained Variance for PCA9: 0.0239	Cumulative Explained Variance up to PCA9: 0.7301
Explained Variance for PCA10: 0.0219	Cumulative Explained Variance up to PCA10: 0.7520
Explained Variance for PCA11: 0.0196	Cumulative Explained Variance up to PCA11: 0.7716
Explained Variance for PCA12: 0.0188	Cumulative Explained Variance up to PCA12: 0.7904
Explained Variance for PCA13: 0.0179	Cumulative Explained Variance up to PCA13: 0.8083
Explained Variance for PCA14: 0.0167	Cumulative Explained Variance up to PCA14: 0.8250
Explained Variance for PCA15: 0.0160	Cumulative Explained Variance up to PCA15: 0.8410
Explained Variance for PCA16: 0.0152	Cumulative Explained Variance up to PCA16: 0.8562
Explained Variance for PCA17: 0.0142	Cumulative Explained Variance up to PCA17: 0.8705
Explained Variance for PCA18: 0.0139	Cumulative Explained Variance up to PCA18: 0.8843
Explained Variance for PCA19: 0.0133	Cumulative Explained Variance up to PCA19: 0.8976
Explained Variance for PCA20: 0.0125	Cumulative Explained Variance up to PCA20: 0.9100
Explained Variance for PCA21: 0.0110	Cumulative Explained Variance up to PCA21: 0.9210
Explained Variance for PCA22: 0.0100	Cumulative Explained Variance up to PCA22: 0.9310
Explained Variance for PCA23: 0.0089	Cumulative Explained Variance up to PCA23: 0.9399
Explained Variance for PCA24: 0.0085	Cumulative Explained Variance up to PCA24: 0.9484
Explained Variance for PCA25: 0.0082	Cumulative Explained Variance up to PCA25: 0.9566

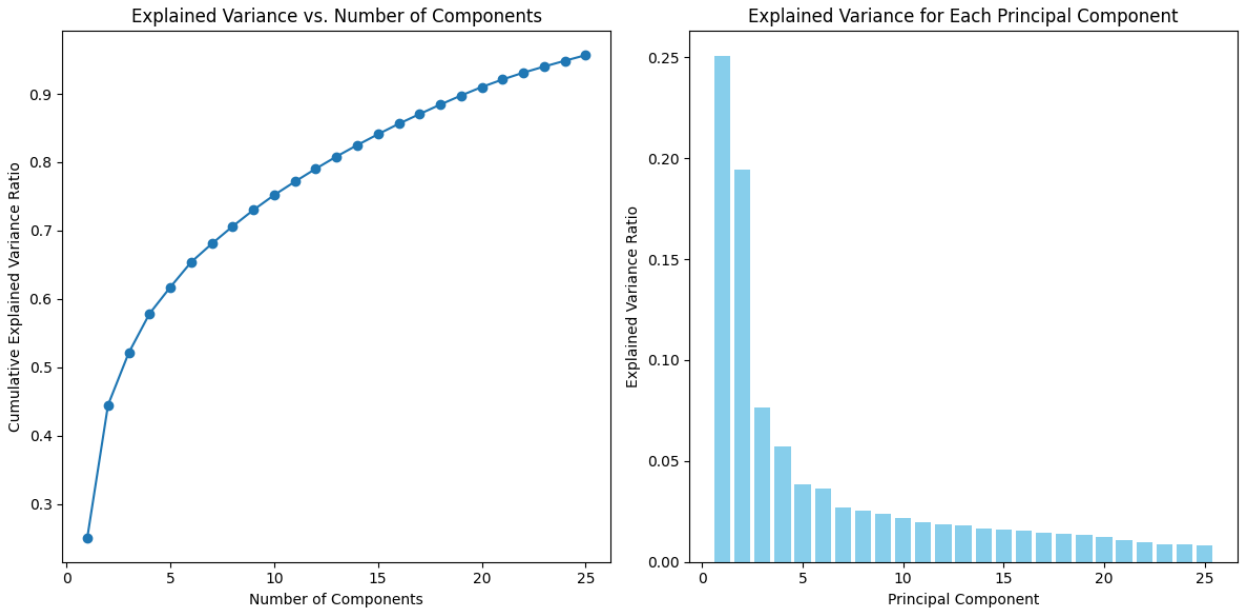

Figure 2 Plot of the Explained Variance and Cumulative Variance vs number of components of Training Dataset

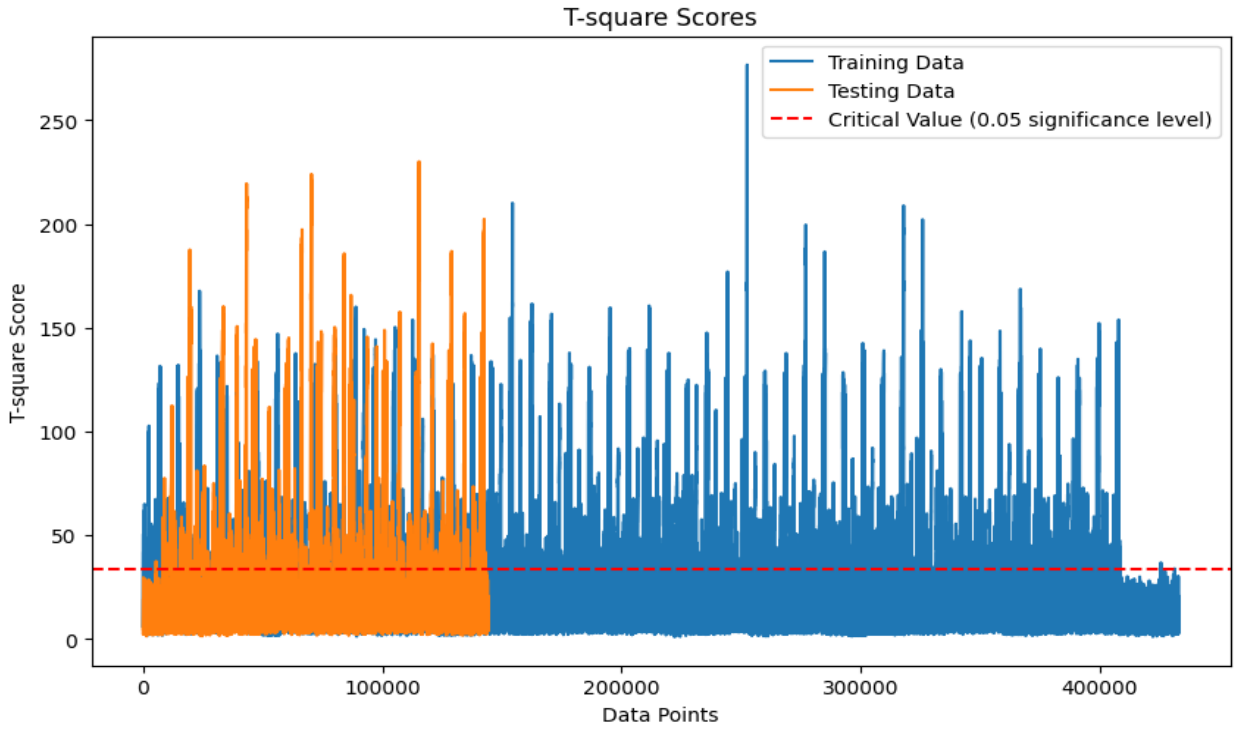

Figure 4 Hoteling T-square lot of Traing Datset

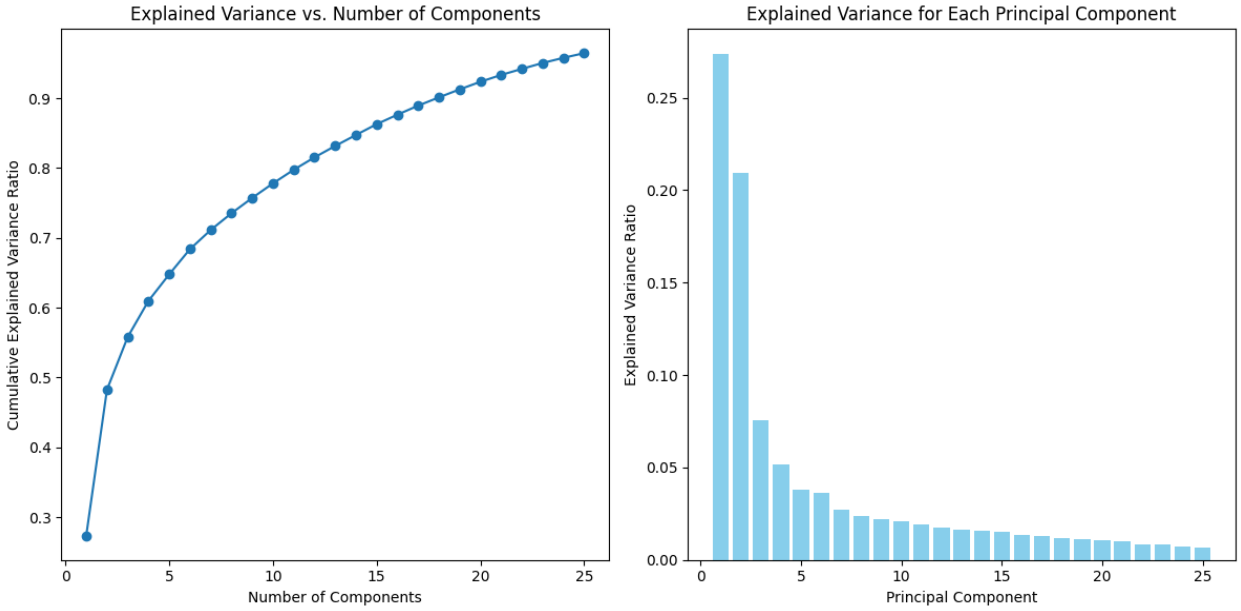

Figure 3 Plot of the Explained Variance and Cumulative Variance vs number of components of Testing Dataset

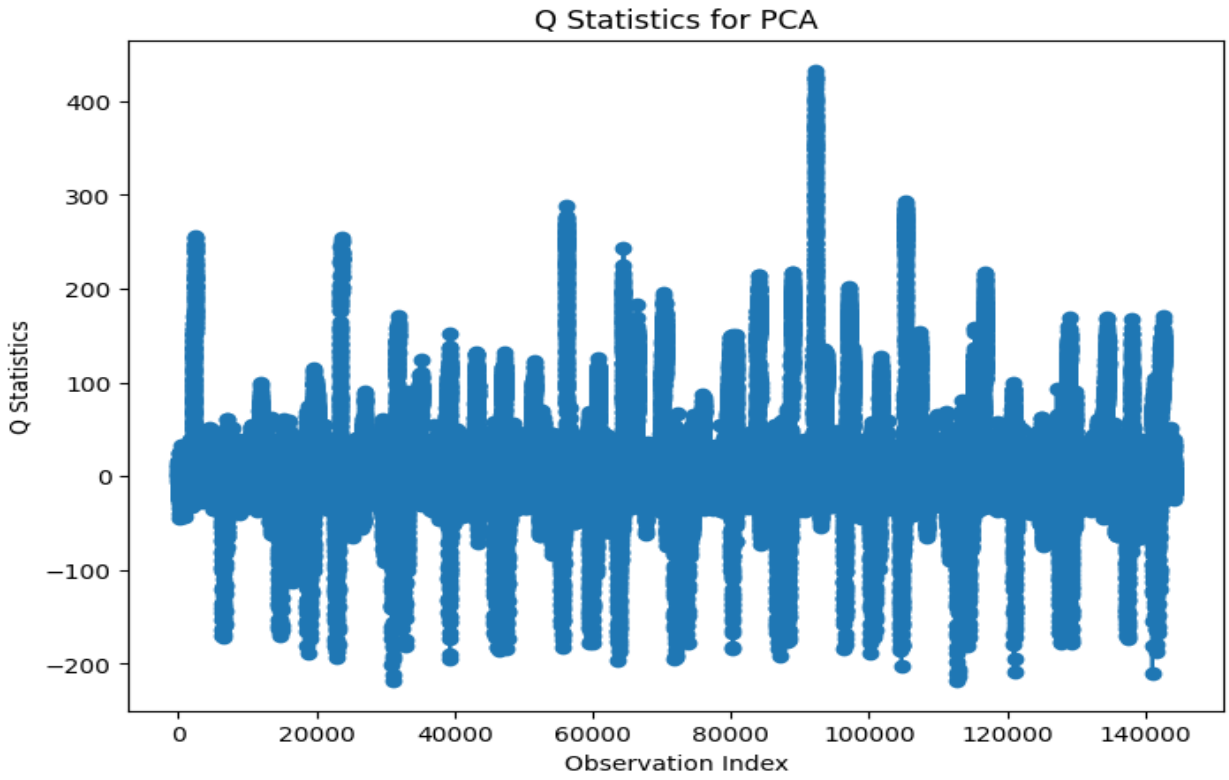

Figure 5 Q statistics plot of Training Datset

4.3 Logistic Regression

Logistic regression is classical linear combination of the input features and pass through logistic function call sigmoid to produce probability of each class. Based on the regression model output it will predict the test features datapoint and assign the positive class.

$Z = b + W_1.X_1 + W_2X_2 + W_3X_3 + \ldots\ldots + WnXn$

1) INPUT dataset: -

   X_train = NTFT500_train_features
   y_train = NTFT500_train_lable
   X_test = NTFTS50_test_features
   y_test = NTFTS50_test_lable

2) Loss Function:
   $J(\theta) = -1/m \sum_{i=1}^{m} [y_{(i)} \cdot \log(h\theta(x(i))) + (1 - y(i)) \cdot \log(1 - h\theta(x(i)))]$
   $h\theta(x(i))$ *is the porbabilty function and* $y(i)$*is the true lable*
   The value of the log loss function = 2.8008

3) Number of Parameters: number of features +1 => 53 + 1 => 54

4) Output of the model: -

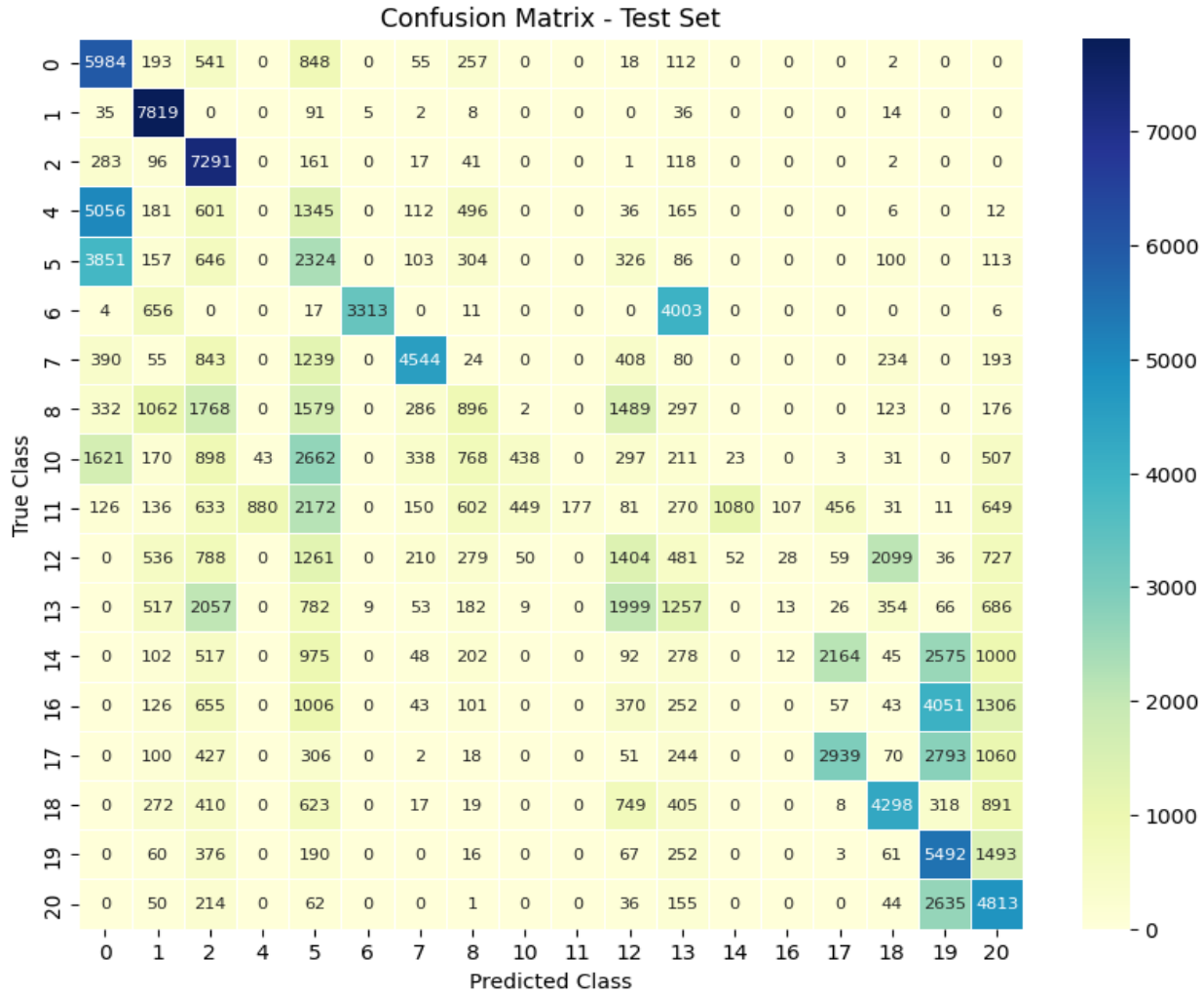

Figure 6 Confusion Matrix of Test Set of LR

Table 6 Classification Report of Logistic Regression model

	precision	recall	f1-score	support
0	0.34	0.75	0.47	8010
1	0.64	0.98	0.77	8010
2	0.39	0.91	0.55	8010
4	0.00	0.00	0.00	8010
5	0.13	0.29	0.18	8010
6	1.00	0.41	0.58	8010
7	0.76	0.57	0.65	8010
8	0.21	0.11	0.15	8010
10	0.46	0.05	0.10	8010

11	1.00	0.02	0.04	8010
12	0.19	0.18	0.18	8010
13	0.14	0.16	0.15	8010
14	0.00	0.00	0.00	8010
16	0.00	0.00	0.00	8010
17	0.51	0.37	0.43	8010
18	0.57	0.54	0.55	8010
19	0.31	0.69	0.42	8010
20	0.35	0.60	0.44	8010
accuracy			0.37	144180
macro avg	0.39	0.37	0.31	144180
weighted avg	0.39	0.37	0.31	144180

4.4 KNN (K-Nearest Neighbor)

KNN is the machine learning language which use both regression and classification techniques. Basic flow of this method is the it takes test dataset to train the model. Calculate the Euclidean Distance between the data point of the feature. Identify the k-nearest neighbors from the data point based on the distance. Then it classification by assigning the label that is nearer the K-Nearest neighbor. Then it predicts the average the value of the assign target variable via regression

Euclidean Distance: $d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{k} \sqrt{(P_i - Q_i)^2}$

Classification = $\hat{y}(x) = \sum_{i=1}^{n} I(y_i = c)$ where
$\hat{y}(x)$ *is the predicted class* , I *is indicator function*

Regression = $\hat{y}(x) = \frac{1}{k}\sum_{i=1}^{k} y_i$

1) INPUT dataset: -

   X_train = NTFT500_train_features
   y_train = NTFT500_train_lable
   X_test = NTFTS50_test_features
   y_test = NTFTS50_test_lable

2) Loss Function: - KNN is unsupervised model so there is no loss function.
3) Hyperparameter: - For the KNN k number of neighbor is the hyperparameter which is 18 in the project

4) Output of the model: Overall Accuracy = 0.62

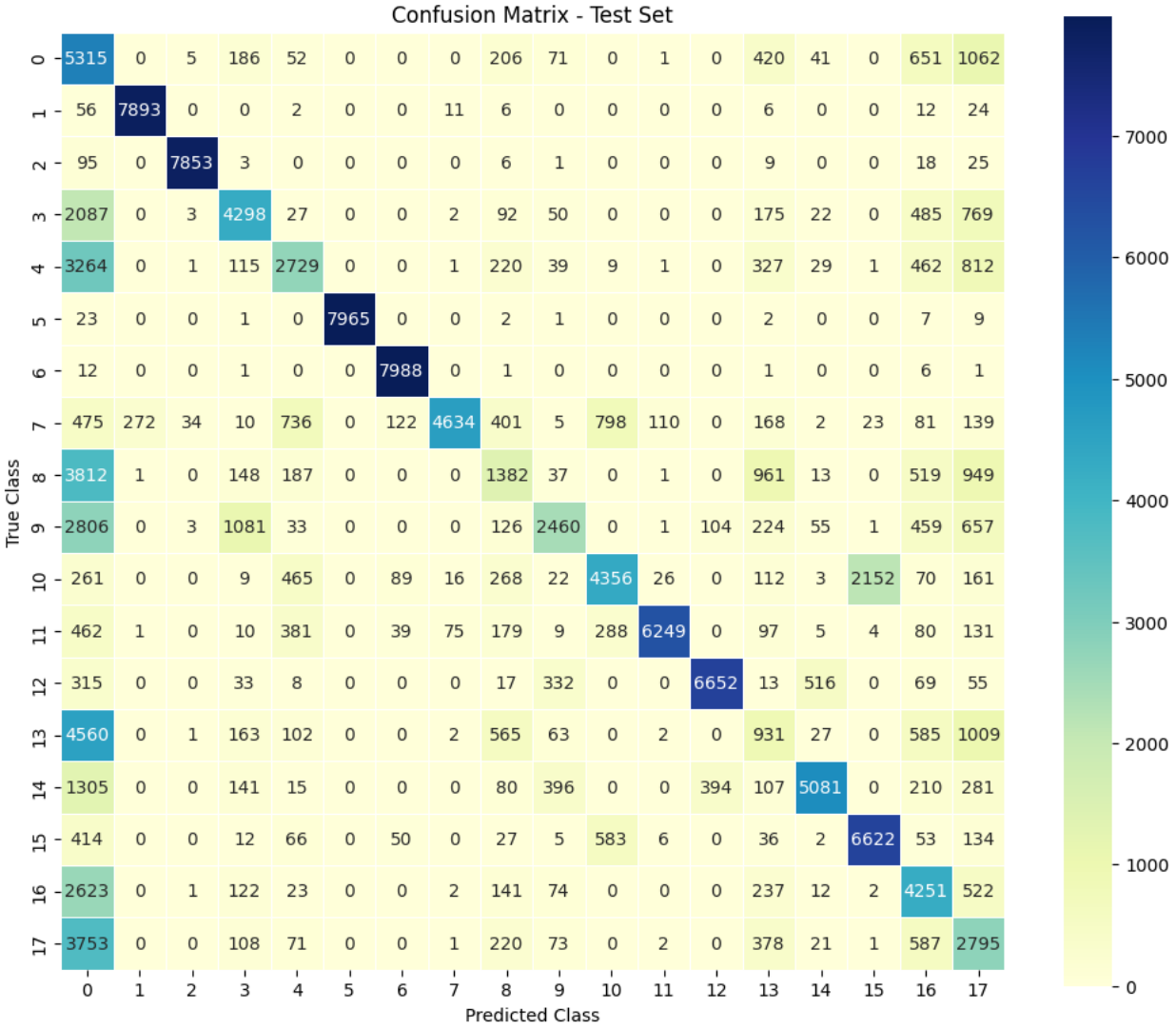

Figure 7 Confusion Matrix of Test Dataset KNN model

Table 7Classification Report of Test set of KNN model

	precision	recall	f1-score	support
0	0.17	0.66	0.27	8010
1	0.97	0.99	0.98	8010
2	0.99	0.98	0.99	8010
4	0.67	0.54	0.59	8010
5	0.56	0.34	0.42	8010
6	1.00	0.99	1.00	8010

7	0.96	1.00	0.98	8010
8	0.98	0.58	0.73	8010
10	0.35	0.17	0.23	8010
11	0.68	0.31	0.42	8010
12	0.72	0.54	0.62	8010
13	0.98	0.78	0.87	8010
14	0.93	0.83	0.88	8010
16	0.22	0.12	0.15	8010
17	0.87	0.63	0.73	8010
18	0.75	0.83	0.79	8010
19	0.49	0.53	0.51	8010
20	0.29	0.35	0.32	8010
Accuracy			0.62	144180
macro avg	0.70	0.62	0.64	144180
weighted avg	0.70	0.62	0.64	144180

4.5 Neural network (NN): -

NN is also machine learning techniques. A NN receive more than one input, perform the weighted sum of all the inputs, applied the activation function rectified linear unit (ReLU) and produce the output. For the validation of the model 20% of the training features set used via default validation split function.

1) INPUT dataset: -

   X_train = NTFT500_train_features
   y_train = NTFT500_train_lable
   X_test = NTFTS50_test_features
   y_test = NTFTS50_test_lable

2) Loss Function: - Loss Function of the NN is defined as
   J (W, B) = $J\,(W,B) = \frac{1}{m}\sum_{i=1}^{m} L\left(y^{(i)},\hat{y}^{(i)}\right)$ where W is weight of the inputs, B represent bias, m is training example.

3) Parameter: - Total 6357 number of parameters are trained and update during the model training.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	3584
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 21)	693

==========
Total params: 6357
Trainable params: 6357
Non-trainable params: 0

4) Output of the model: Overall Accuracy = 0.88

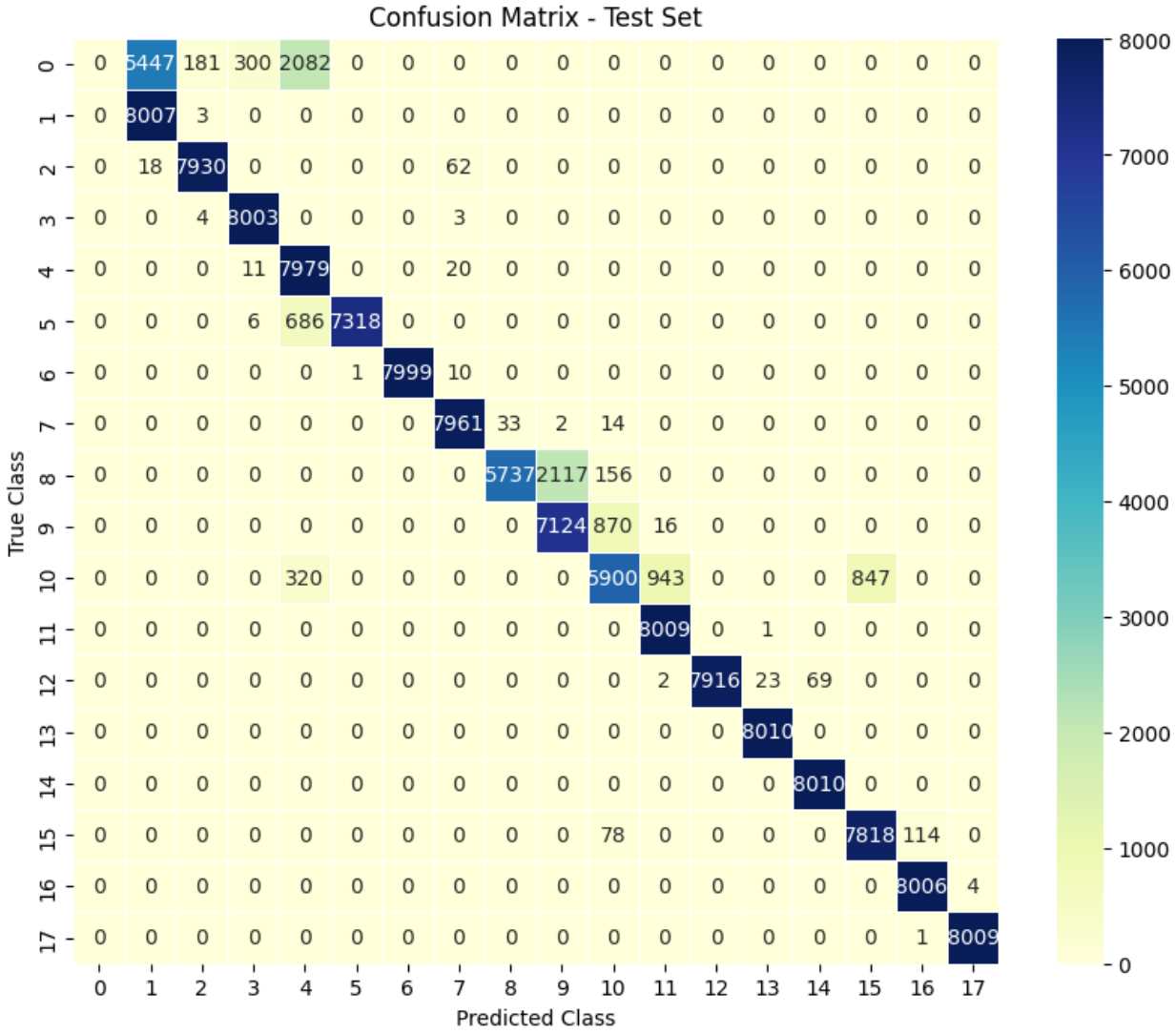

Figure 8 Confusion Matrix of Test dataset of NN

Table 8 Confusion Matrix of Test Set of NN model

	precision	recall	f1-score	support
0	0.00	0.00	0.00	8010
1	0.59	1.00	0.75	8010

2	0.98	0.99	0.98	8010
4	0.96	1.00	0.98	8010
5	0.72	1.00	0.84	8010
6	1.00	0.91	0.95	8010
7	1.00	1.00	1.00	8010
8	0.99	0.99	0.99	8010
10	0.99	0.72	0.83	8010
11	0.77	0.89	0.83	8010
12	0.84	0.74	0.79	8010
13	0.89	1.00	0.94	8010
14	1.00	0.99	0.99	8010
16	1.00	1.00	1.00	8010
17	0.99	1.00	1.00	8010
18	0.90	0.98	0.94	8010
19	0.99	1.00	0.99	8010
20	1.00	1.00	1.00	8010
accuracy			0.90	144180
macro avg	0.87	0.90	0.88	144180
weighted avg	0.87	0.90	0.88	144180

4.6 KNN + NN (K-Nearest Neighbor supported by NN)

This is novelty approach used in this project based on the similar ideas available in literatures (Eslamloueyan, 2011). The basic flow of this model is that we gave training dataset features as input to the K-Means clustering for the training data and assign each sample to one of the $n_{clusters}$ clusters. The same clustering model is then used to predict cluster memberships for the test data. The same clustering model is then used to predict cluster memberships for the test data. Now we will give inputs to the NN model. Train dataset is the training feature matrix and output of the train (cluster_membership_train)[2] is the array of cluster assignments obtained from K-Means. The cluster memberships are used as labels for training a neural network. The model is trained on the training data, using cluster assignments as labels. The combined model is evaluated using classification metrics. For the validation of the model 20% of the feature training set via default validation split function.

1) INPUT dataset: -

X_train = NTFT500_train_features
y_train = NTFT500_train_lable
X_test = NTFTS50_test_features

[2] For more details regarding the cluster membership train, go to appending section page no.34 where all the variables are defined.

y_test = NTFTS50_test_lable

2) Loss: - Graphs are attached with below output of the model part.

3) Parameter: - Total 6258 number of parameters are trained and update during the model training. Please find below mentioned model summary.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 64)	3584
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 18)	594

Total params: 6258
Trainable params: 6258
Non-trainable params: 0

4) Output of the model: Overall Accuracy = 0.98

Table 9 Classification Matrix of Test set of KNN+NN model

Fault	precision	recall	f1-score	support
0	0.99	0.98	0.99	21446
1	0.99	0.87	0.93	3277
2	0.97	1.00	0.98	3869
3	0.99	1.00	0.99	2934
4	1.00	1.00	1.00	6702
5	0.94	0.97	0.96	4650
6	1.00	1.00	1.00	3674
7	0.99	0.99	0.99	23442
8	0.96	1.00	0.98	23313
9	0.88	1.00	0.94	2378
10	0.99	0.95	0.97	5216
11	0.99	0.99	0.99	712
12	0.97	0.93	0.95	4507
13	1.00	0.97	0.99	8098
14	0.99	0.98	0.98	1037
15	0.96	0.97	0.96	4531
16	0.98	0.98	0.98	21848
17	0.99	0.98	0.99	2546
accuracy			0.98	144180
macro avg	0.98	0.97	0.98	144180
weighted avg	0.98	0.98	0.98	144180

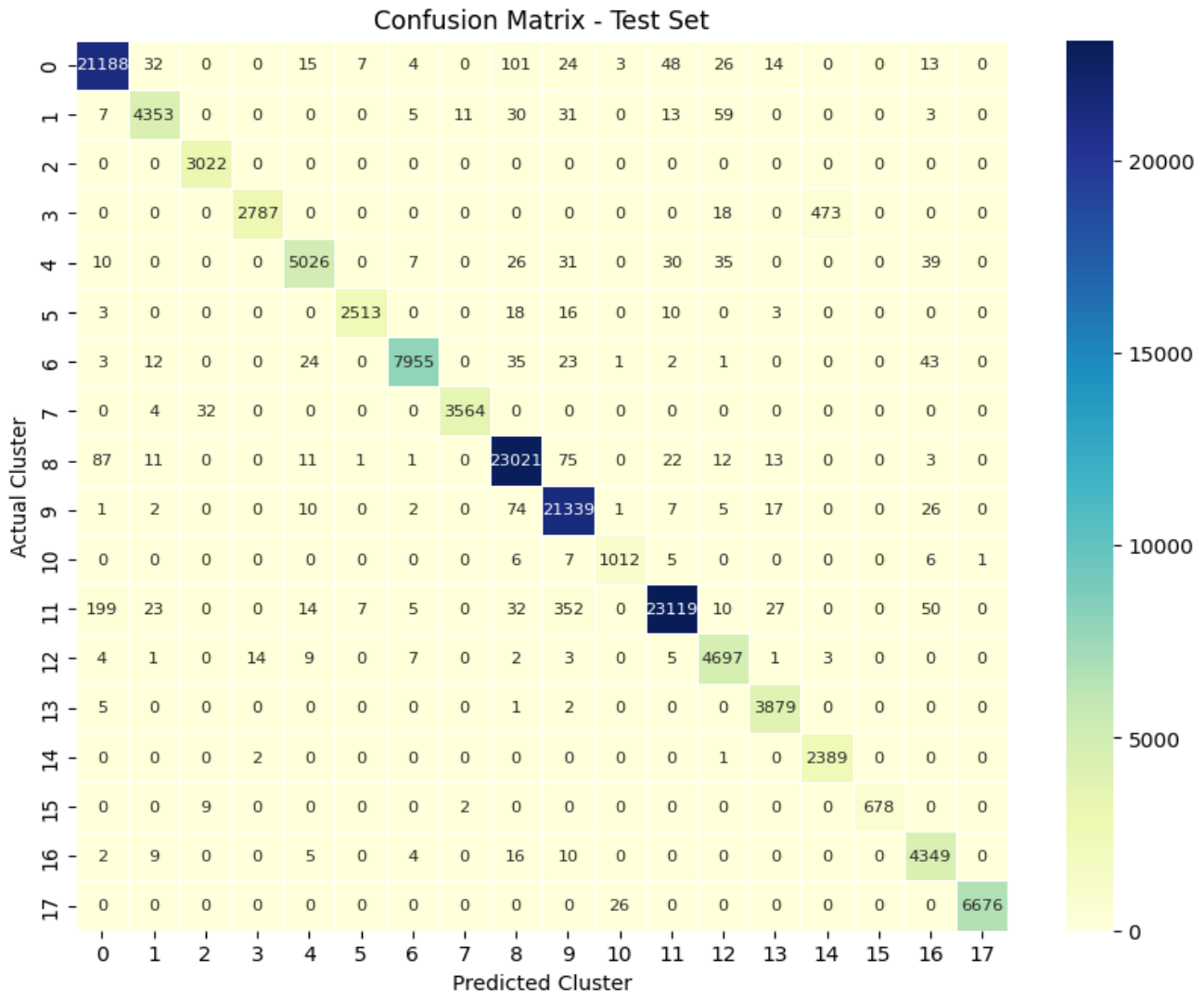

Figure 9 confusion matric of Test dataset of KNN+NN model.

5. Result and Discussion: -

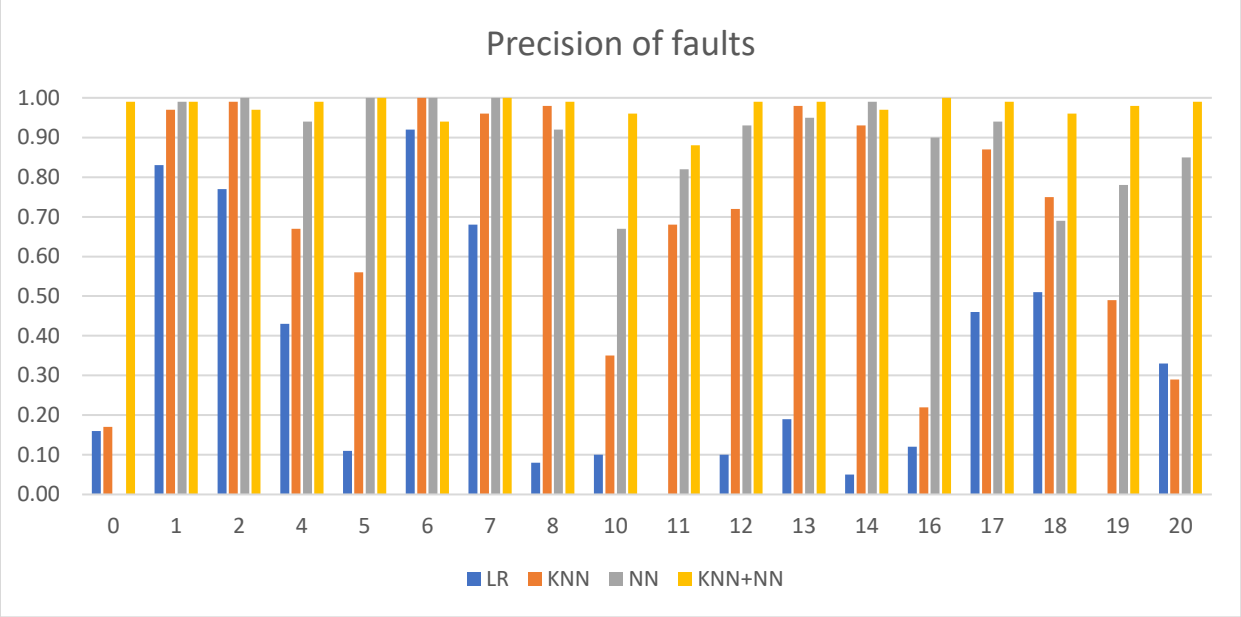

Figure 10 Precision matrix of LR, KNN, NN, KNN+NN

Table 10 Heatmap of Precision Matrix of the each faut by LR, KNN, NN and KNN+NN model

Precision Matrix of each fault				
Fault	**LR**	**KNN**	**NN**	**KNN+NN**
0	0.16	0.17	0.00	0.99
1	0.83	0.97	0.99	0.99
2	0.77	0.99	1.00	0.97
4	0.43	0.67	0.94	0.99
5	0.11	0.56	1.00	1.00
6	0.92	1.00	1.00	0.94
7	0.68	0.96	1.00	1.00
8	0.08	0.98	0.92	0.99
10	0.10	0.35	0.67	0.96
11	0.00	0.68	0.82	0.88
12	0.10	0.72	0.93	0.99
13	0.19	0.98	0.95	0.99

14	0.05	0.93	0.99	0.97
16	0.12	0.22	0.90	1.00
17	0.46	0.87	0.94	0.99
18	0.51	0.75	0.69	0.96
19	0.00	0.49	0.78	0.98
20	0.33	0.29	0.85	0.99

Table 11 Comparison of the Accuracy of LR, KNN, NN and KNN+NN

	LR	**KNN**	**NN**	**KNN+NN**
Test Accuracy Percentage	36%	62%	89%	98%

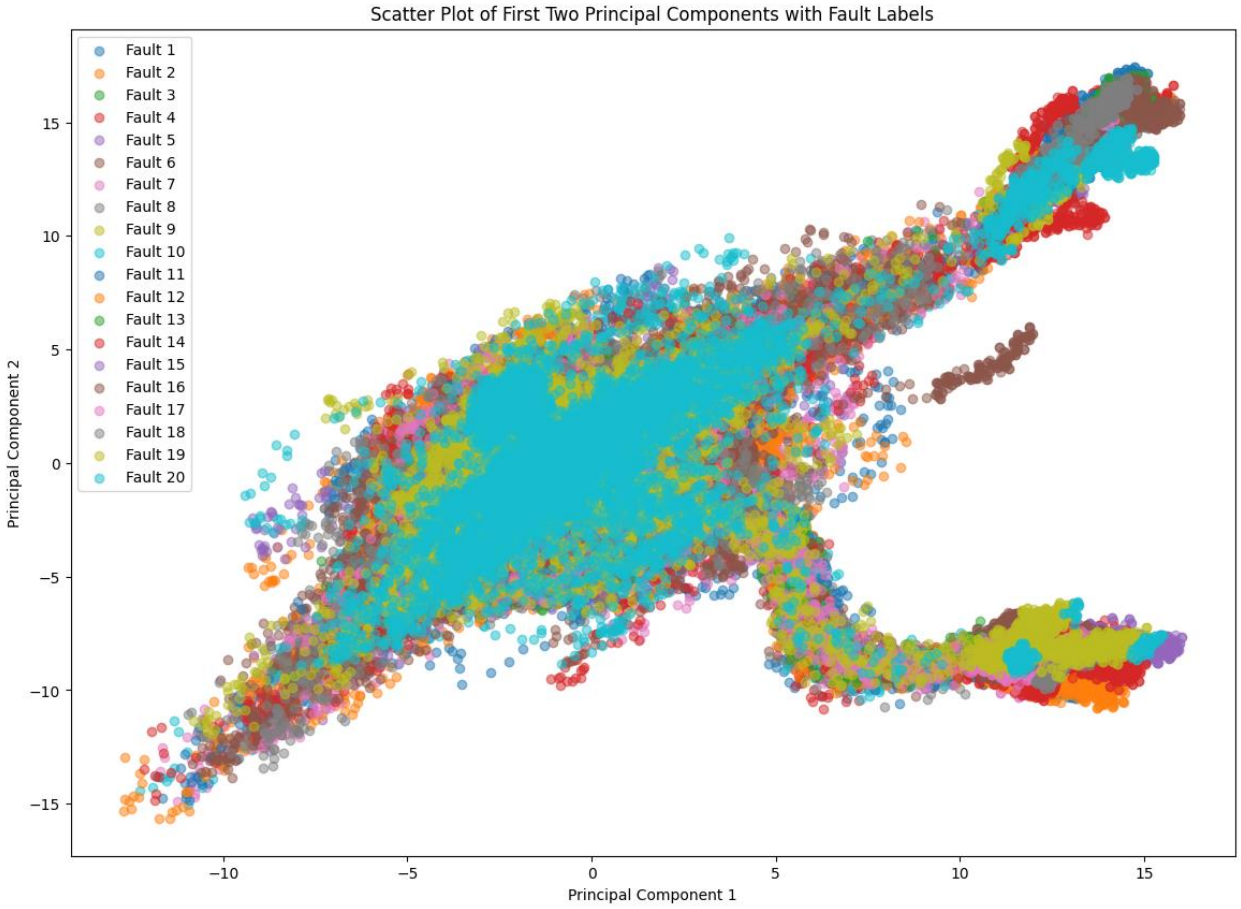

Figure 11 Principal component 2 vs Principal component 1 for all faults

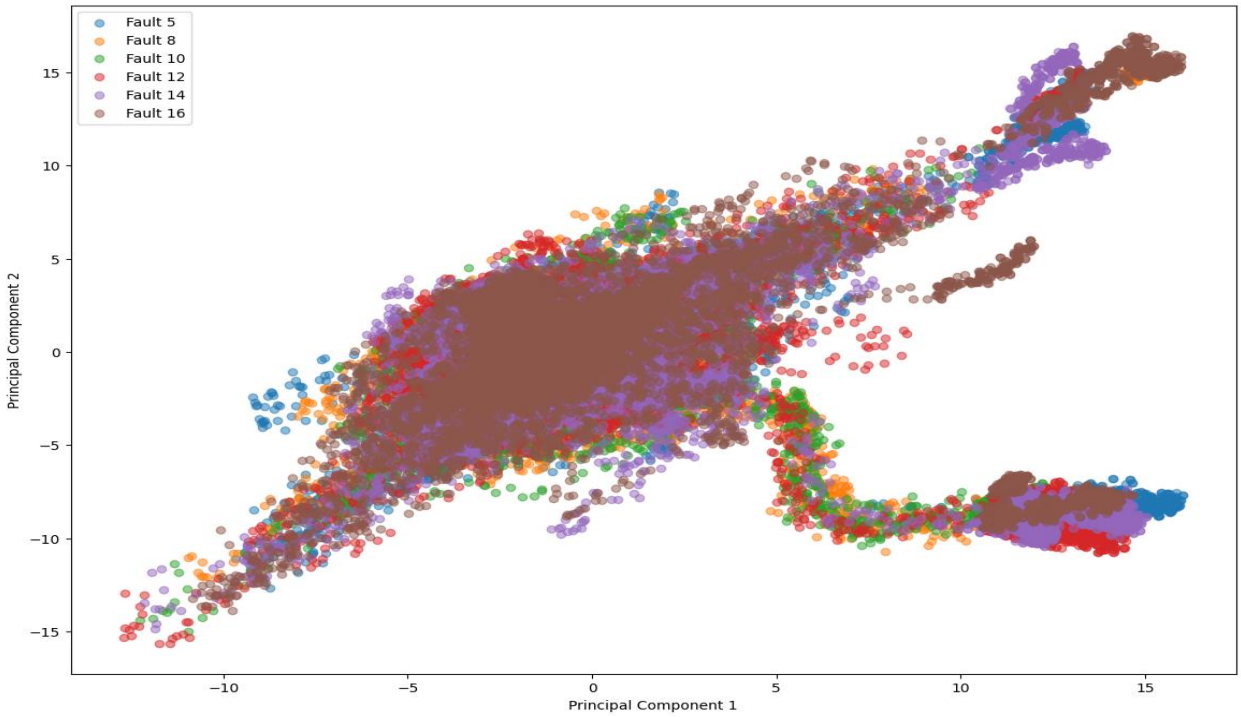

Figure 13 PCA2 vs PCA1 of Fault no. 5,8,10,12,14&16

- From the Heatmap of precision matrix as well as the accuracy table of the model, it can easily see that KNN+NN is the best method among the all-other model used in the project. Even though, alone NN model has comparatively good accuracy, it fails to detect normal operating region (Fault zero [0] the datapoint), which make KNN+NN superior from all the models used.

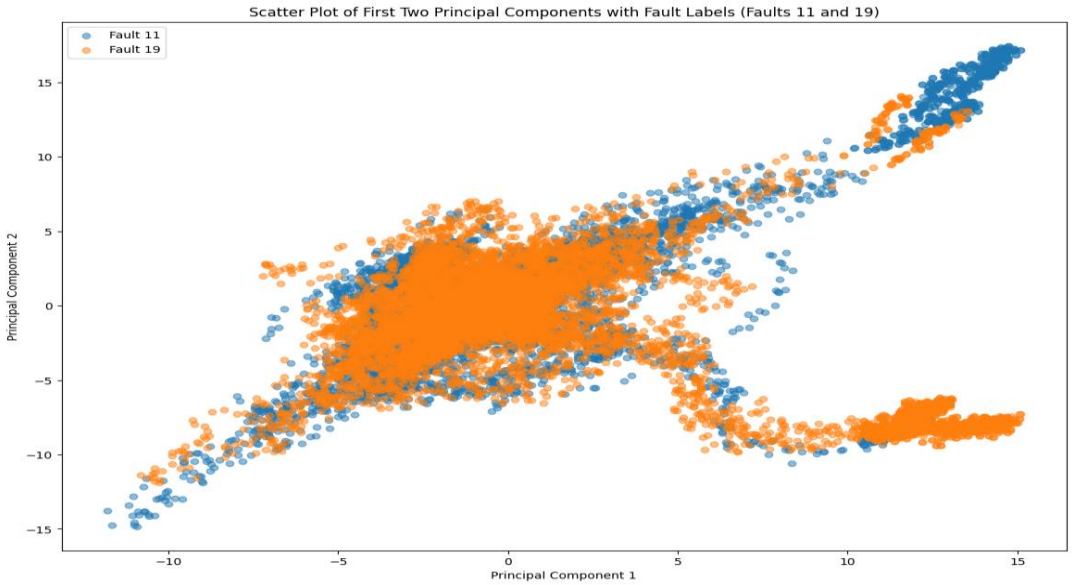

Figure 12 PCA1 vs PCA2 for Fault no. 11 & 19

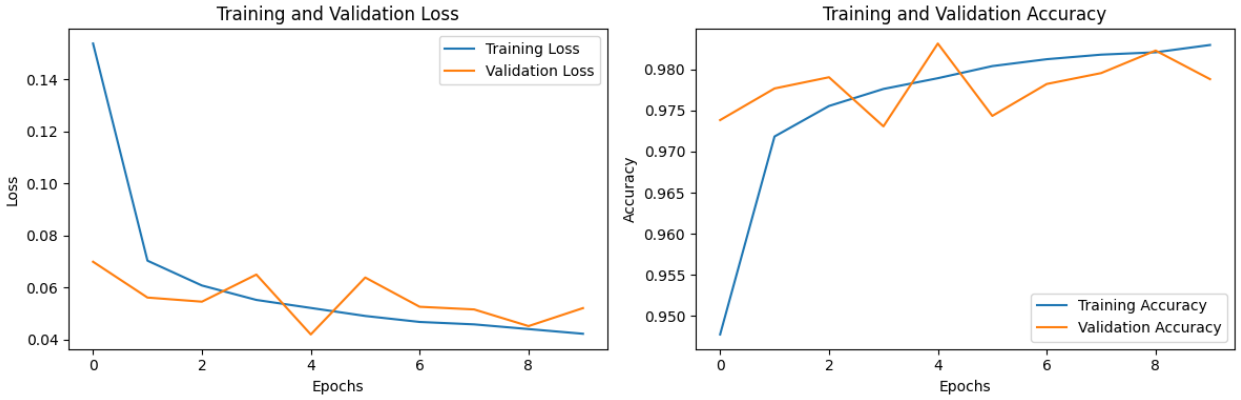

Figure 14 Training and Validation Loss and Accuracy of KNN+NN model

- For Logistic Regression (LR) model, it has lowest fault precision among all. Fault no 11 and 19 is not detect by the LR. Furthermore, fault number 5,8,10,12,14 also have lower precision than 15%. To detect the reason behind the LR failure, scatterplot was drawn between PC1 and PC2. From the figure it can interpreted that the data is highly non-linear. Therefore, the prediction of the data label from the test feature is not good. The scatter plot of PCA component of the fault 11 and 19 also plot for better understating of the data non-linearity withing the datapoint. The same reason for less precision to identification of fault 5,8,10,12,14. Hence, LR is not a good model for the TEP dataset we have taken.
- For the KNN – From the precision matrix it can easily say that KNN overall prediction of the faulty dataset is good though it can predict (0.17) the zero fault (norma condition) as compare as the other faults which is least among all the precision of datapoint by the models. Fault number of 10 and 20 has the lower precision whereas the fault 6 has highest precision 1 with re-call of 0.99
- NN model working very effective compare to the KNN. However, it still failing to detect the normal class (zero [0] faults) which is the worst than KNN predict the zero faults. Other than that, it was better than KNN. Lowest precision for detecting the faults is Fault 18 is the 0.69. Fault number 2,5,6.7 has precision of 1.00 and the fault number 1,4, 12,13,14,16,17 has precision more than 0.90. Therefore, it can conclude almost all the faults accurate other than the normal operating condition.
- To overcome the KNN and NN limitation to detect the normal operating condition and improve the precision of the faults that we are lacking in these models, KNN assisted by NN (KNN+NN) model is used. It not only detects the normal condition dataset but also have exception precision with almost 1.00. Almost all the faults have the precision more than the 0.90 except the fault number 11. Even fault number 11 has also had the precision value of the 0.88 which is greater than KNN and NN both model precision of 0.67 and 0.82 respectively. From the figure 10. it can see that loss function can be reduce to 0.05 within the 10 Epochs. This can be reduced to less than 0.01 with more Epochs like 30. However, this can not only lead to computation expensiveness but also the overfitting of the data as accuracy is not changing significantly. Therefore, 10 Epochs are chosen for the model.

Table 12 Comparison of the test accuracy result of Literature available

Sr	Model	SimulationRun	Faults excluded for FDD	Accuracy Percentage
1	KNN + NN	50	3, 9, 15	98.0%
2.	LSTM (Pangun Park, 2019) Ilong short-term memory)	500	3, 9, 15	91.9%
3.	DCNN (Pangun Park, 2019) (Deep Convolutional Neural Network)	500	3, 9, 15	76.4%

- In this report, accuracy result was compared from the publish literature by *Pangun et.* (Pangun Park, 2019) with similar dataset which also excluded the dataset of fault 3, 9, 15. *Pangun et.* (Pangun Park, 2019) *all* worked with all the simulationRun (500/500) where as this project only include first 50 simulationRun (50/500). *Pangun et. all* (Pangun Park, 2019)used the LSTM (long short-term memory) model and DCNN (Deep Convolutional Neural Network) for fault detection. KNN+NN model got 21.4% more accuracy than the DCNN model and 6.1% more accurate than LSTM. Hence, proposed approach KNN+NN model is superior[3] with limited dataset of TEP process and fault exclusion of 3,9,15.

[3] Here word superior is only refer with assumption that we only take limited dataset to train and test the model therefore it has significantly good accuracy and precision of the fault dataset. If we change the sample size of the training dataset and test dataset it might change the result.

6. References

Downs, J. J. (1993). A plant-wide industrial process control problem. *Computers and Chemical Engineering,*, 17(3), 245-255. doi:https://doi.org/10.1016/0098-1354(93)80018-I

Eslamloueyan, R. (2011). Designing a hierarchical neural network based on fuzzy clustering for fault diagnosis of the Tennessee–Eastman process. 1407-1415. doi:doi:10.1016/j.asoc.2010.04.012

Pangun Park, P. D. (2019). Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network. *sensors*, 4612-4630. doi:doi:10.3390/s19214612

Seongmin Heo, J. H. (2019). Statistical Process Monitoring of the Tennessee Eastman Process Using Parallel Autoassociative Neural Networks and a Large Dataset. *Processes 7*, 411.

Zhang, Y. (2009,). Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM. *Chemical Engineering Science,*, 801-811. doi:https://doi.org/10.1016/j.ces.2008.10.012.

7. Appendix[4]

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import pairwise_distances
from scipy.stats import chi2
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Load The Fault Test and Train dataset
path = "/kaggle/input/tep-dataset-1-5-gb-csv/faulty_testing.csv"
FTS  =  pd.read_csv (path)
path1 = "/kaggle/input/tep-dataset-1-5-gb-csv/faulty_training.csv"
FTN  =  pd.read_csv (path1)

# Normal Tets and Training Dataset
path11 = "/kaggle/input/tep-dataset-1-5-gb-csv/normal_training.csv"
NTN  =  pd.read_csv (path11)
path12 = "/kaggle/input/tep-dataset-1-5-gb-csv/normal_testing.csv"
NTS  =  pd.read_csv (path12)

missing_data = FTS.isnull().sum()
#missing_data = FTN.isnull().sum()
#missing_data = NTN.isnull().sum()
#missing_data = NTS.isnull().sum()
print("Missing Data:")
print(missing_data)

df_NTT = pd.DataFrame(FTN)
df_FTT = pd.DataFrame(NTN)
NTFTN_500 = pd.concat([df_NTT, df_FTT], axis=0, ignore_index=True)
NTFTN_500 = NTFTN_500[(NTFTN_500['simulationRun'] >= 1) & (NTFTN_500['simulationRun'] <= 50)]
NTFTN_500 = NTFTN_500[NTFTN_500['faultNumber'] != 3]
NTFTN_500 = NTFTN_500[NTFTN_500['faultNumber'] != 9]
NTFTN_500 = NTFTN_500[NTFTN_500['faultNumber'] != 15]

NTFTN_500 = NTFTN_500[(NTFTN_500['sample'] >= 20) & (NTFTN_500['sample'] <= 500)]
NTFT500_train_features = NTFTN_500.iloc[:, 3:55]
NTFT500_train_lable = NTFTN_500.iloc[:, 0]
#print(NTFTN_500.shape)

df_NTTS = pd.DataFrame(NTS)
```

[4] All the results are attached in the form of graph or tables. Kaggle notebook is used for code run and all the analysis. You can visit the and see the result over there under name of Rtr Abhi Chauhan.

```
df_FTTS = pd.DataFrame(FTS)
NTFTS_500 = pd.concat([df_NTTS, df_FTTS], axis=0, ignore_index=True)
NTS50 = NTFTS_500[(NTFTS_500['simulationRun'] >= 1) & (NTFTS_500['simulationRun'] <= 10)]
NTS50 = NTS50[NTS50['faultNumber'] != 3]
NTS50 = NTS50[NTS50['faultNumber'] != 9]
NTS50 = NTS50[NTS50['faultNumber'] != 15]

NTFTS50 = NTS50[(NTS50['sample'] >= 160) & (NTS50['sample'] <= 961)]
NTFTS50_test_features = NTFTS50.iloc[:, 3:55]
NTFTS50_test_lable = NTFTS50.iloc[:, 0]
NTFTS50.faultNumber.value_counts()
#print(NTS50.shape)
```

Applying PCA on the Training Dataset

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(NTFT500_train_features)

n_components = 25

pca_train = PCA(n_components=n_components)
pca_result_train = pca_train.fit_transform(scaled_data)

# Plot the Explained Variance for Each Principal Component
plt.subplot(1, 2, 2)
plt.bar(range(1, n_components + 1), pca_train.explained_variance_ratio_, color='skyblue')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance for Each Principal Component')
plt.tight_layout()
plt.show()
# Ploting the Cumulative Explained Variance vs. Number of Components
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, n_components + 1), np.cumsum(pca_train.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance vs. Number of Components')

# Print the explained variance of each principal component
for i, ratio in enumerate(pca_train.explained_variance_ratio_, 1):
    print(f"Explained Variance for PCA{i}: {ratio:.4f}")
explained_variance_ratio = pca_train.explained_variance_ratio_
# Print the cumulative explained variance of each principal component
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
for i, cumulative_ratio in enumerate(cumulative_explained_variance, 1):
    print(f"Cumulative Explained Variance up to PCA{i}: {cumulative_ratio:.4f}")

# Print the Reconstruction Error after PCA
reconstructed_data = pca_train.inverse_transform(pca_result_train)
mse = np.mean(np.square(scaled_data - reconstructed_data))

print(f"Reconstruction Error (Mean Squared Error): {mse:.4f}")
```

Make a scatterplot for first two principal component of every faults

```
plt.figure(figsize=(14, 10))
for fault_number in range(1, 21):  # Assuming 20 fault numbers
    indices = NTFTN_500[NTFTN_500['faultNumber'] == fault_number].index

    # Check if indices are within the valid range
    valid_indices = indices[indices < pca_result_train.shape[0]]

    plt.scatter(
        pca_result_train[valid_indices, 0],
        pca_result_train[valid_indices, 1],
        label=f'Fault {fault_number}',
        alpha=0.5
    )

plt.title('Scatter Plot of First Two Principal Components with Fault Labels')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Find and Plot Q-statistics

```
def calculate_q_statistics(pca, scaled_data, reconstructed_data):
    # Calculate the residuals
    residuals = scaled_data - reconstructed_data[:scaled_data.shape[0]]  # Adjust the shape of
reconstructed_data

    # Calculate the loading matrix (transform of principal components)
    loading_matrix = pca.components_.T

    # Calculate the Q statistics for each observation
    q_statistics = np.sum(residuals ** 2 @ loading_matrix, axis=1)

    return q_statistics
q_stats = calculate_q_statistics(pca_train, scaled_data, reconstructed_data)
# Plot the Q statistics
plt.figure(figsize=(8, 6))
plt.plot(q_stats, marker='o')
plt.xlabel('Observation Index')
plt.ylabel('Q Statistics')
plt.title('Q Statistics for PCA')
plt.show()
```

Find and plot T-square statistics

```
# Calculate T-square scores
t_square_train = np.sum((X_train_pca / np.std(X_train_pca, axis=0)) ** 2, axis=1)
t_square_test = np.sum((X_test_pca / np.std(X_train_pca, axis=0)) ** 2, axis=1)
```

```
# Set significance level for Hotelling's T-square test
alpha = 0.05

# Calculate critical value for Hotelling's T-square test
df = n_components
critical_value = chi2.ppf(1 - alpha, df)

plt.figure(figsize=(10, 6))
plt.plot(t_square_train, label='Training Data')
plt.plot(t_square_test, label='Testing Data')
plt.axhline(critical_value, color='r', linestyle='--', label=f'Critical Value ({alpha} significance level)')
plt.title('T-square Scores')
plt.xlabel('Data Points')
plt.ylabel('T-square Score')
plt.legend()
plt.show()
print(f'Critical Value for T-square Test: {critical_value:.4f}')
```

TEST DATASET PCA

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(NTFTS50_test_features)

n_components = 25

pca_test = PCA(n_components=n_components)
pca_result_test = pca_test.fit_transform(scaled_data)

# Plot the Explained Variance for Each Principal Component
plt.subplot(1, 2, 2)
plt.bar(range(1, n_components + 1), pca_test.explained_variance_ratio_, color='skyblue')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance for Each Principal Component')
plt.tight_layout()
plt.show()

# Plot the Cumulative Explained Variance vs. Number of Components
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, n_components + 1), np.cumsum(pca_test.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance vs. Number of Components')

# Print the explained variance of each principal component
for i, ratio in enumerate(pca_test.explained_variance_ratio_, 1):
    print(f"Explained Variance for PCA{i}: {ratio:.4f}")
```

Logistic Regression Code

```
# -------Defining the train & test data with features and lables---------#
X_train = NTFT500_train_features
```

```
y_train = NTFT500_train_lable
X_test = NTFTS50_test_features
y_test = NTFTS50_test_lable

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

logreg_model = LogisticRegression(random_state=42)

logreg_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logreg_model.predict(X_test)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# -------Plot the confusion matrix---------#

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="YlGnBu", cbar=True, linewidths=.5, square=True,
        xticklabels=np.unique(y_pred), yticklabels=np.unique(y_pred),
        annot_kws={"size": 8})
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

KNN Classifier

```
# -------Defining the train & test data with features and lables---------#
X_train = NTFT500_train_features
y_train = NTFT500_train_lable
X_test = NTFTS50_test_features
y_test = NTFTS50_test_lable

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=21)

# Fit the model on the training data
```

```
knn_classifier.fit(X_train_scaled, y_train)

# Predictions on the test set
y_test_pred = knn_classifier.predict(X_test_scaled)

# Evaluate the model on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
conf_matrix_test = confusion_matrix(y_test, y_test_pred)
class_report_test = classification_report(y_test, y_test_pred)

print("\nTest Accuracy:", accuracy_test)
print("Confusion Matrix (Test):")
print(conf_matrix_test)
print("Classification Report (Test):")
print(class_report_test)

# -------Plot the confusion matrix---------#

plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix_test, annot=True, fmt="d", cmap="YlGnBu", cbar=True, linewidths=.5,
square=True)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

Neural Network (NN)

```
# -------Defining the train & test data with features and lables---------#
X_train = NTFT500_train_features
y_train = NTFT500_train_lable
X_test = NTFTS50_test_features
y_test = NTFTS50_test_lable

# Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert labels to one-hot encoding
num_classes = 18 # 17 faults + 0 fault(Normal)
y_train_one_hot = tf.keras.utils.to_categorical(y_train, num_classes=num_classes)
y_test_one_hot = tf.keras.utils.to_categorical(y_test, num_classes=num_classes)

# Build the neural network model
model = tf.keras.Sequential([
   tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
   tf.keras.layers.Dense(32, activation='relu'),
   tf.keras.layers.Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
# Train the model
model.fit(X_train_scaled, y_train_one_hot, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
y_test_pred_one_hot = model.predict(X_test_scaled)
y_test_pred_classes = np.argmax(y_test_pred_one_hot, axis=1)

# Evaluate the model's performance
accuracy_test = accuracy_score(y_test, y_test_pred_classes)
conf_matrix_test = confusion_matrix(y_test, y_test_pred_classes)
class_report_test = classification_report(y_test, y_test_pred_classes)


print("\nTest Accuracy:", accuracy_test)
print("Classification Report (Test):")
print(class_report_test)

# -------Plot the confusion matrix---------#

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix_test, annot=True, fmt="d", cmap="YlGnBu", cbar=True, linewidths=.5,
square=True)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

KNN+NN (with Plot of loss and accuracy)

```
# -------Defining the train & test data with features and lables---------#
X_train = NTFT500_train_features
y_train = NTFT500_train_lable
X_test = NTFTS50_test_features
y_test = NTFTS50_test_lable

# Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

n_clusters = 18
kmeans = KMeans(n_clusters=n_clusters)
cluster_membership_train = kmeans.fit_predict(X_train_scaled)
cluster_membership_test = kmeans.predict(X_test_scaled)

# Build the neural network model
model = tf.keras.Sequential([
   tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
   tf.keras.layers.Dense(32, activation='relu'),
```

```
    tf.keras.layers.Dense(n_clusters, activation='softmax')  # Output layer with n_clusters nodes
])

# Compile the model with Categorical Cross-Entropy loss
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train the model and capture the history
history = model.fit(X_train_scaled, cluster_membership_train, epochs=10, batch_size=32,
validation_split=0.2)

# Access and print the loss and accuracy values
loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
accuracy_values = history.history['accuracy']
val_accuracy_values = history.history['val_accuracy']

print("Training Loss Values:")
print(loss_values)
print("Validation Loss Values:")
print(val_loss_values)
print("Training Accuracy Values:")
print(accuracy_values)
print("Validation Accuracy Values:")
print(val_accuracy_values)

# Plot the loss values
plt.figure(figsize=(12, 4))

# Plot Training and Validation Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot Training and Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate the model on the test set
y_test_pred_clusters = np.argmax(model.predict(X_test_scaled), axis=1)

# Generate a classification report and confusion matrix
print("Classification Report:")
```

```
print(classification_report(cluster_membership_test, y_test_pred_clusters))


# -------Plot the confusion matrix---------#

conf_matrix = confusion_matrix(cluster_membership_test, y_test_pred_clusters)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="YlGnBu", cbar=True, linewidths=.5, square=True,
        xticklabels=np.unique(y_test_pred_clusters), yticklabels=np.unique(y_test_pred_clusters),
        annot_kws={"size": 8})
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted Cluster')
plt.ylabel('True Cluster')
plt.show()
```