ELG 5142 Ubiquitous Sensing for Smart Cities

Assignment 3

Group23_HW3

Overview

The aim of this project that we get familiar with anomaly detection in time series and know how to apply anomaly detection models like: SVM, KNN, PCA and DBSCAN.

1. Import libraries

We used Pyod library for SVM, PCA and KNN and we used Sklearn for DBSCAN.

```
from pyod.models.pca import PCA
from pyod.models.knn import KNN
from pyod.models.ocsvm import OCSVM
import matplotlib.pyplot as plt

import pandas as pd, numpy as np, re
from sklearn.cluster import DBSCAN
from sklearn.manifold import TSNE
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import accuracy_score
```

2. Our Functions

```python
#-----------------------------------Functions-----------------------------------

# Function to read the Dataset
def readDataSet(DataSet_name, Sheet_Name):
    Extension = re.findall('((.csv)|(.xls)|(.xlsx))', DataSet_name)
    Extension = str(Extension)
    if '.csv' in Extension:
        Extension = '.csv'
    elif '.xls' in Extension:
        Extension = '.xls'
    elif '.xlsx' in Extension:
        Extension = '.xlsx'
    if Extension == ('.xls' or '.xlsx'):
        DataFrame = pd.read_excel(DataSet_name, sheet_name = Sheet_Name)
    elif Extension == '.csv':
        DataFrame = pd.read_csv(DataSet_name)
    return DataFrame

# to separate the points that belong for each class
def GetListOfClasses(numberOfClasses, DataSet, TargetColumn):
        ls = [None] * numberOfClasses
        for i in range(0,numberOfClasses):
            ls[i] = DataSet.loc[DataSet[TargetColumn] == i]
        return ls

# Plot DataPoints
def PlotDataPoints(numberOfClasses, ListOFClasses, XLabel, Ylabel , Label0,label1 , S, Title):
    colorsOptions = ['#FF0000','#8941FF','blue','#00FF0F',' #FF00AE','#000000','#0B9397','#1B49CD','#11AF29','#560078','#ED036A']
    MarkersOptions = ['h','*','v','^','D','x','X','P','H','d']
    colors = colorsOptions[0:numberOfClasses]
    Markers = MarkersOptions[0:numberOfClasses]

    plt.scatter(x = ListOFClasses[0].iloc[:, 0:1], y = ListOFClasses[0].iloc[:, 1:2], c=colors[0], marker = Markers[0], s=S, label = Label0)
    plt.scatter(x = ListOFClasses[1].iloc[:, 0:1], y = ListOFClasses[1].iloc[:, 1:2], c=colors[1], marker = Markers[1], s=S, label = label1)

    plt.xlabel(XLabel, fontsize = 15)
    plt.ylabel(Ylabel, fontsize = 15)
    plt.title(Title)
    plt.legend()
    return plt

# T_Sne
def T_SNE(X):
    return TSNE(n_components=2).fit_transform(X)

# to generate Confusion Matrix
def ConfusionMatrix(Y_Actual, Y_Pred):
    CF = confusion_matrix(Y_Actual, Y_Pred)
    return CF
```

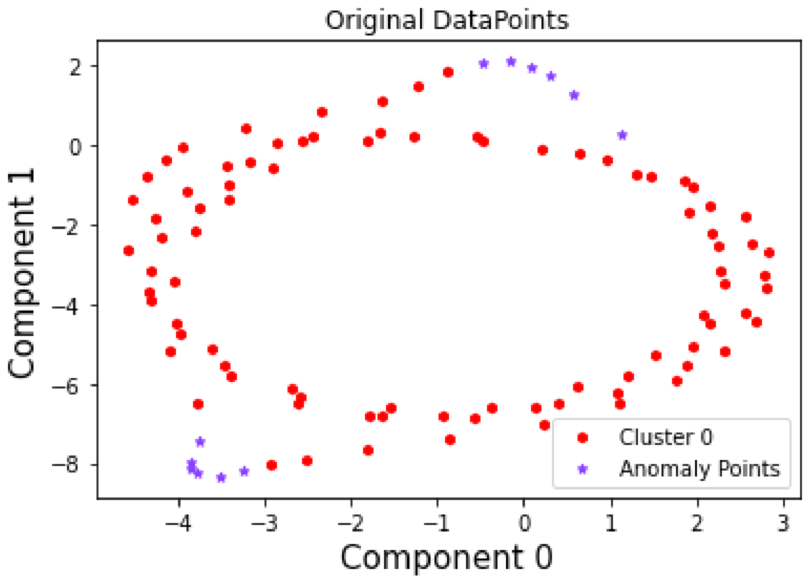
```python
# to Plot Confusion Matrix
def PLOT_ConfusionMatrix(CF,Title):
    sns.heatmap(CF, annot=True, fmt='d')
    plt.title(Title, fontsize = 15)
    plt.xlabel('Predicted', fontsize = 15)
    plt.ylabel('Actual', fontsize = 15)
    return plt.show()

# Accuracy on Test
def AccuracyTest(Y_Actual, Y_Pred):
    return accuracy_score(Y_Actual, Y_Pred) * 100

# to Plot TimeSeries
def TimeSeries_plt(Df):
    plt.plot(Df.iloc[:,0], Df.iloc[:,1], color='#00A093', label = 'Follower_measure_x_follower')
    plt.plot(Df.iloc[:,0], Df.iloc[:,2], color='#C94000', label = 'Follower_measure_y_follower')
    plt.plot(Df.iloc[:,0], Df.iloc[:,3], color='#B4007A', label = 'Leader_measure_x_leader')
    plt.plot(Df.iloc[:,0], Df.iloc[:,4], color='#00B400', label = 'Leader_measure_y_leader')
    plt.legend(fontsize=7)
    plt.xlabel('Time', fontsize = 15)
    plt.ylabel('4 Features', fontsize = 15)
    return plt

# Plot Anomaly Points
def PltDetected_Points(df, y, Title):
    data = pd.concat([pd.DataFrame(df), pd.DataFrame(y)],axis=1 , ignore_index = True).astype(float)
    data = data[data.iloc[:,5]==1]
    plt.scatter(x= data.iloc[:,0],y= data.iloc[:,1],color ='#00A093', marker = 'o', s=15)
    plt.scatter(x= data.iloc[:,0],y= data.iloc[:,2],color ='#C94000', marker = 'o', s=15)
    plt.scatter(x= data.iloc[:,0],y= data.iloc[:,3],color ='#B4007A', marker = 'o', s=15)
    plt.scatter(x= data.iloc[:,0],y= data.iloc[:,4],color ='#00B400', marker = 'o', s=15)
    plt.title(Title)
    return plt.show()

# to Plot Bar Chart
def Bar(x, y, W, lsOFcolor , Xlabel, Ylabel , Title, Label):
    plt.bar(x, y, W, color = lsOFcolor, label=Label)
    plt.xlabel(Xlabel)
    plt.ylabel(Ylabel)
    plt.legend(loc = 3)
    plt.title(Title)
    return plt
```

3. Read the Dataset

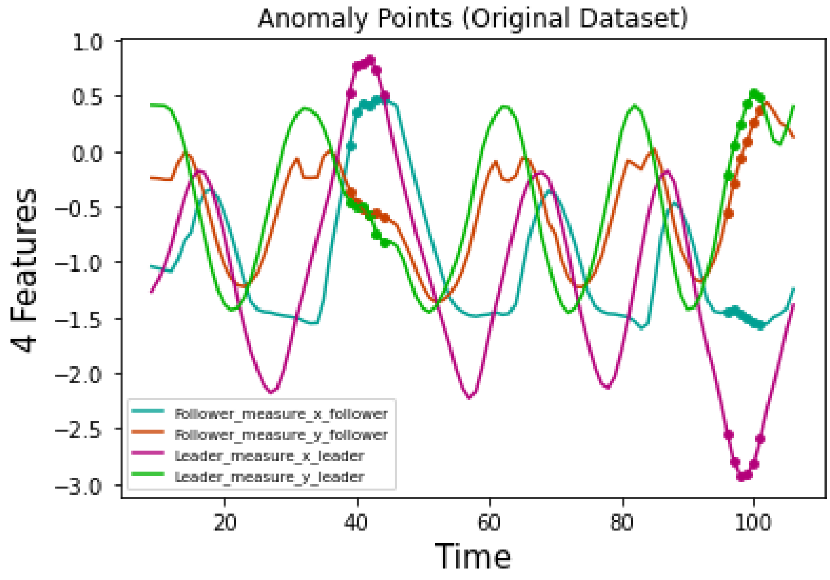
```
#-----------------------------------MAIN-----------------------------------
# Read The Dataset
FullData = readDataSet('Dataset_to_be_used_in_performance_comparison.csv', 'Dataset_to_be_used_in_performan')
X = FullData.iloc[:,1:5]
Y = FullData.iloc[:,[5]]
```

4. Plot the original datapoints Time Series (with anomaly points) and T_sne

```
#--------------------Plot Original Data--------------------
XTsne = T_SNE(X)
XTsne = pd.concat([pd.DataFrame(XTsne), pd.DataFrame(Y)],axis=1 , ignore_index = True).astype(float)
Original = GetListOfClasses(2, XTsne,  pd.DataFrame(XTsne).columns[2])
PlotDataPoints(2, Original, 'Component 0','Component 1' ,'Cluster 0','Anomaly Points' ,20 , 'Original DataPoints').show()

#--------------------Time Series (Anomaly Points)--------------------
TimeSeries = FullData.iloc[:,0:5]
TimeSeries_plt(TimeSeries)
PltDetected_Points(TimeSeries, Y,'Anomaly Points (Original Dataset)')
```



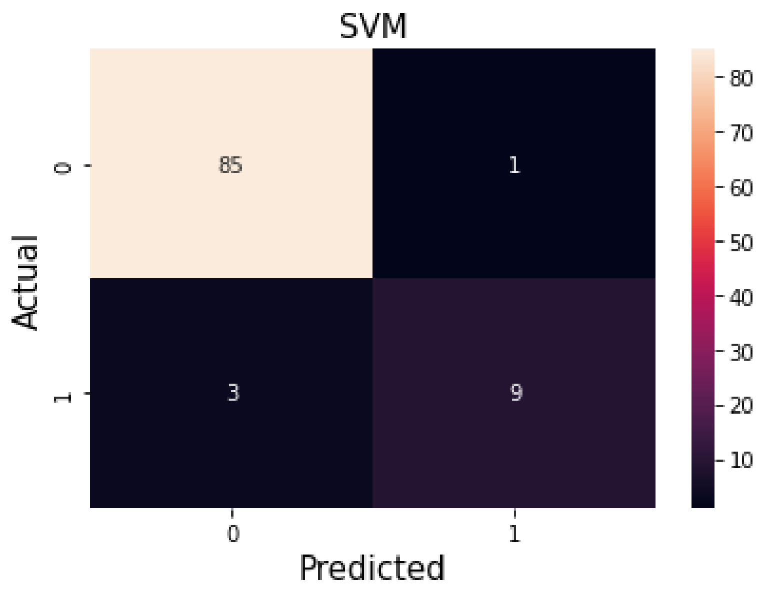
- **T-SNE for original dataset**



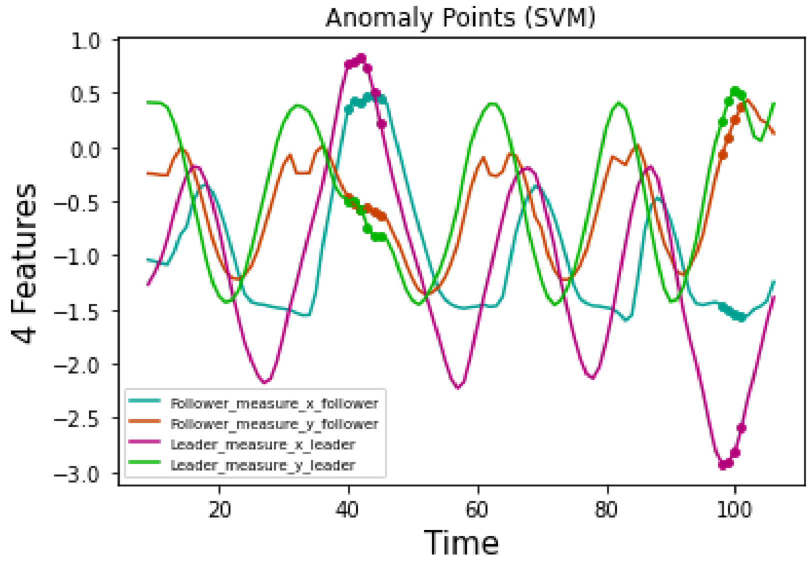
- **Time Series with Anomaly Points for original dataset**

5. SVM

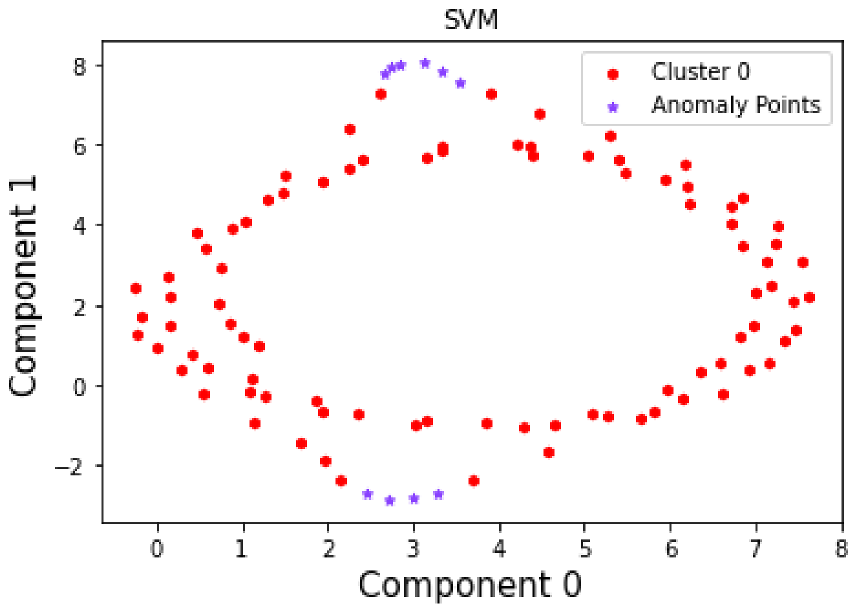
```
#-----------------------------------------SVM-----------------------------------------
#------Train the model and make predictions------
Svm = OCSVM().fit(X)
Svm_Clusters = Svm.predict(X)

# Confusion Matrix and Report
SvmReport = classification_report(Y, Svm_Clusters)
Svm_Cf = ConfusionMatrix(Y, Svm_Clusters)
PLOT_ConfusionMatrix(Svm_Cf,'SVM')

#---------------------SVM Time Series (Anomaly Points)---------------------
TimeSeries_plt(TimeSeries)
PltDetected_Points(TimeSeries, Svm_Clusters,'Anomaly Points (SVM)')

#---------------------Tsne SVM---------------------
XTsne = T_SNE(X)
XTsne = pd.concat([pd.DataFrame(XTsne), pd.DataFrame(Svm_Clusters)],axis=1 , ignore_index = True).astype(float)
SvmLs = GetListOfClasses(2, XTsne,  pd.DataFrame(XTsne).columns[2])
PlotDataPoints(2, SvmLs, 'Component 0','Component 1' ,'Cluster 0','Anomaly Points' ,20 , 'SVM').show()
```



- **SVM Confusion matrix**



- **Time Series with Anomaly Points for SVM**



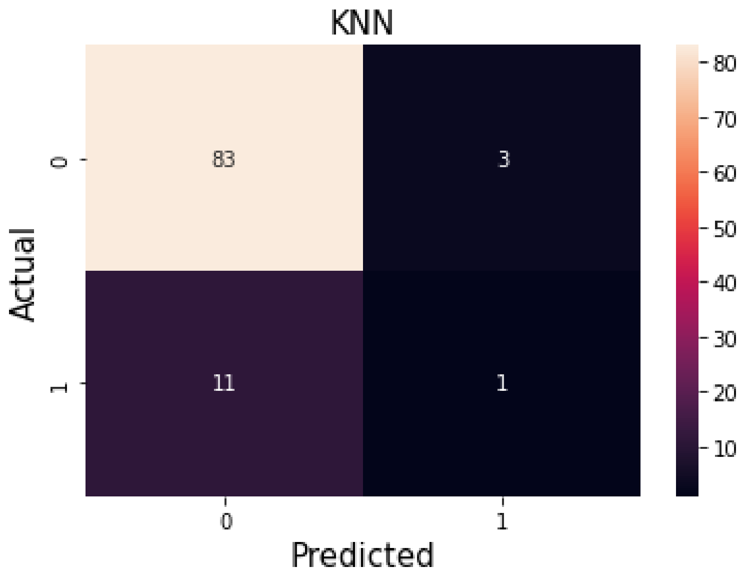
- **T-SNE for SVM**

	precision	recall	f1-score	support
0.0	0.97	0.99	0.98	86
1.0	0.90	0.75	0.82	12
accuracy			0.96	98
macro avg	0.93	0.87	0.90	98
weighted avg	0.96	0.96	0.96	98

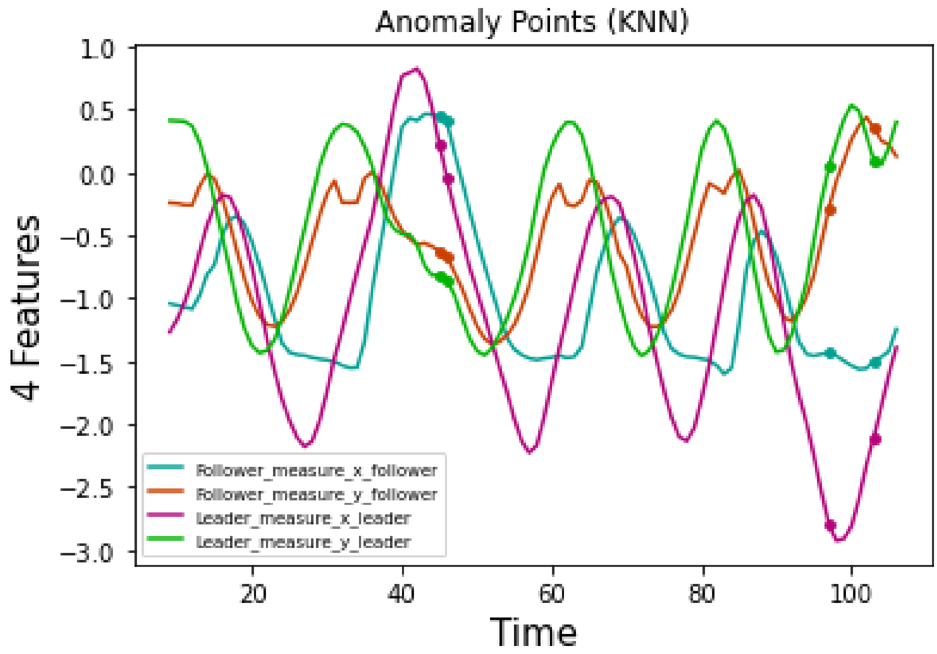
- **Classification Report for SVM**

6. KNN

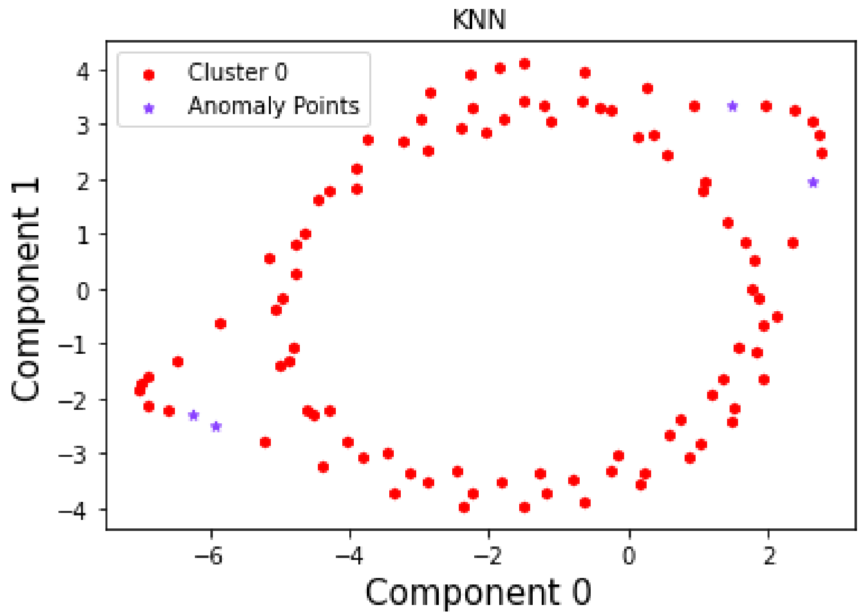
```
#----------------------------------------KNN---------------------------------------
#------Train the model and make predictions------
Knn = KNN().fit(X)
Knn_Clusters = Knn.predict(X)

# Confusion Matrix and Report
KnnReport = classification_report(Y, Knn_Clusters)
Knn_Cf = ConfusionMatrix(Y, Knn_Clusters)
PLOT_ConfusionMatrix(Knn_Cf,'KNN')

#---------------------Knn Time Series (Anomaly Points)---------------------
TimeSeries_plt(TimeSeries)
PltDetected_Points(TimeSeries, Knn_Clusters,'Anomaly Points (KNN)')

#---------------------Tsne KNN--------------------
XTsne = T_SNE(X)
XTsne = pd.concat([pd.DataFrame(XTsne), pd.DataFrame(Knn_Clusters)],axis=1 , ignore_index = True).astype(float)
KnnLs = GetListOfClasses(2, XTsne,  pd.DataFrame(XTsne).columns[2])
PlotDataPoints(2, KnnLs, 'Component 0','Component 1' ,'Cluster 0','Anomaly Points' ,20 , 'KNN').show()
```



- **KNN Confusion matrix**



- **Time Series with Anomaly Points for KNN**



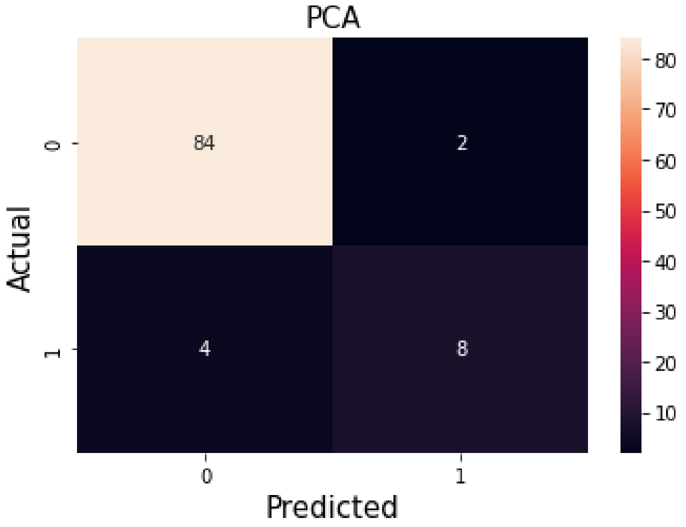
- **T-SNE for KNN**

	precision	recall	f1-score	support
0.0	0.88	0.97	0.92	86
1.0	0.25	0.08	0.12	12
accuracy			0.86	98
macro avg	0.57	0.52	0.52	98
weighted avg	0.81	0.86	0.82	98

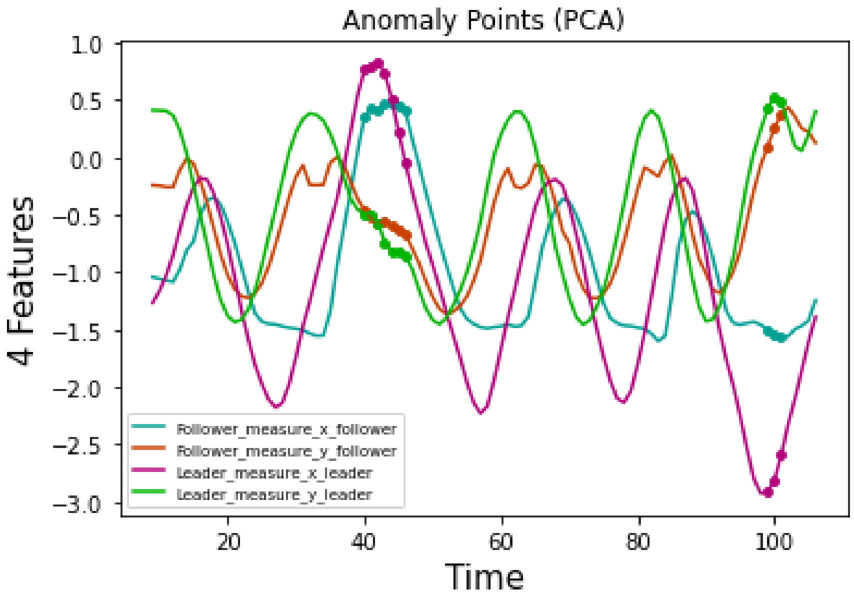
- **Classification Report for KNN**

7. PCA

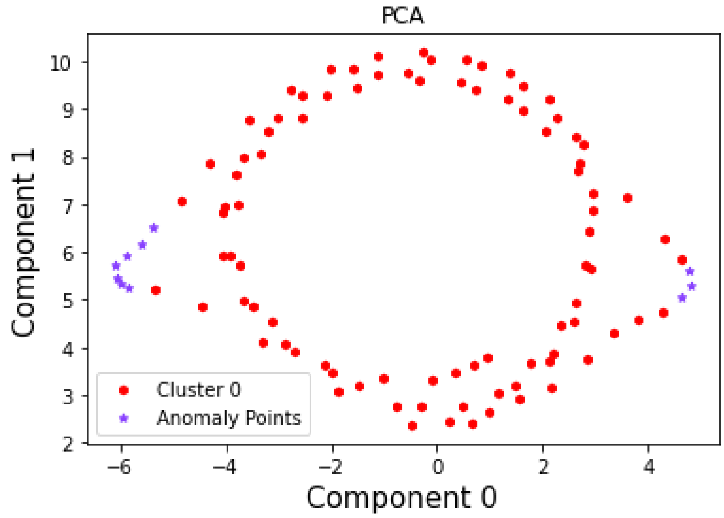
```
#-----------------------------------------PCA-------------------------------------
#------Train the model and make predictions------
Pca = PCA().fit(X)
Pca_Clusters = Pca.predict(X)

# Confusion Matrix and Report
PcaReport = classification_report(Y, Pca_Clusters)
Pca_Cf = ConfusionMatrix(Y, Pca_Clusters)
PLOT_ConfusionMatrix(Pca_Cf,'PCA')

#--------------------Pca Time Series (Anomaly Points)--------------------
TimeSeries_plt(TimeSeries)
PltDetected_Points(TimeSeries, Pca_Clusters,'Anomaly Points (PCA)')

#--------------------Tsne Pca--------------------
XTsne = T_SNE(X)
XTsne = pd.concat([pd.DataFrame(XTsne), pd.DataFrame(Pca_Clusters)],axis=1 , ignore_index = True).astype(float)
PcaLs = GetListOfClasses(2, XTsne,  pd.DataFrame(XTsne).columns[2])
PlotDataPoints(2, PcaLs, 'Component 0','Component 1' ,'Cluster 0','Anomaly Points' ,20 , 'PCA').show()
```



- **PCA Confusion matrix**



- **Time Series with Anomaly Points for PCA**



- **T-SNE for PCA**

	precision	recall	f1-score	support
0.0	0.95	0.98	0.97	86
1.0	0.80	0.67	0.73	12
accuracy			0.94	98
macro avg	0.88	0.82	0.85	98
weighted avg	0.94	0.94	0.94	98

- **Classification Report for PCA**

8. DB_Scan

Here we do a hyperparameter tuning to determine best values for epsilon and MinPoints.

```
#----------------------------------------DBSCAN---------------------------------------------
#-----------DBSCAN hyperparameter tuning -----------
epslist = np.array([0.1,0.2,0.3, 0.4, 0.5, 0.6,0.7])
minPoint = np.array([2,3,4,5,6,7,8,9,10,11,12,13,14,15])

Epsilons =[]
MidPoints = []
DB_acc = []

for i in range(len(epslist)):
    for j in range(len(minPoint)):
        model = DBSCAN(eps=epslist[i], min_samples=minPoint[j])
        DBscan_Clusters = model.fit_predict(X)
        # print('(ep=',epslist[i],' ,mi=', minPoint[j],' )','\n') # to Print the points
        if len(np.unique(DBscan_Clusters)) != 2:
            continue
        else:
            Epsilons.append(epslist[i])
            MidPoints.append(minPoint[j])
            DBscan_Clusters = pd.DataFrame(DBscan_Clusters).replace(-1,1)
            DB_acc.append(AccuracyTest(Y, DBscan_Clusters))

Eps_Min_Acc = pd.concat([pd.DataFrame(Epsilons), pd.DataFrame(MidPoints), pd.DataFrame(DB_acc)],axis=1 , ignore_index = True).astype(float)
Eps_Min_Acc.columns = ['Eps', 'Min', 'Accuracy']
```

And we got concatenated the appended values of epsilons and Minpoints and accuracies to get something like this: -

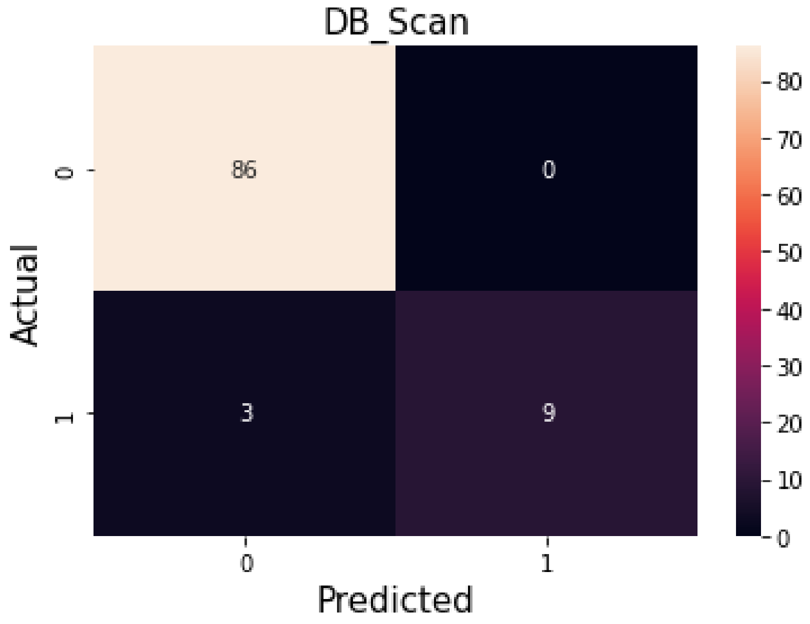
Eps_Min_Acc - DataFrame

Index	Eps	Min	Accuracy
17	0.70000000	10.000…	96.93877551
19	0.70000000	12.000…	96.93877551
20	0.70000000	13.000…	96.93877551
10	0.60000000	8.0000…	95.91836735
11	0.60000000	9.0000…	95.91836735
12	0.60000000	10.000…	95.91836735
18	0.70000000	11.000…	95.91836735
21	0.70000000	14.000…	95.91836735
6	0.50000000	7.0000…	94.89795918
13	0.60000000	11.000…	94.89795918
7	0.50000000	8.0000…	93.87755102
8	0.50000000	9.0000…	93.87755102
14	0.70000000	7.0000…	91.83673469
15	0.70000000	8.0000…	91.83673469

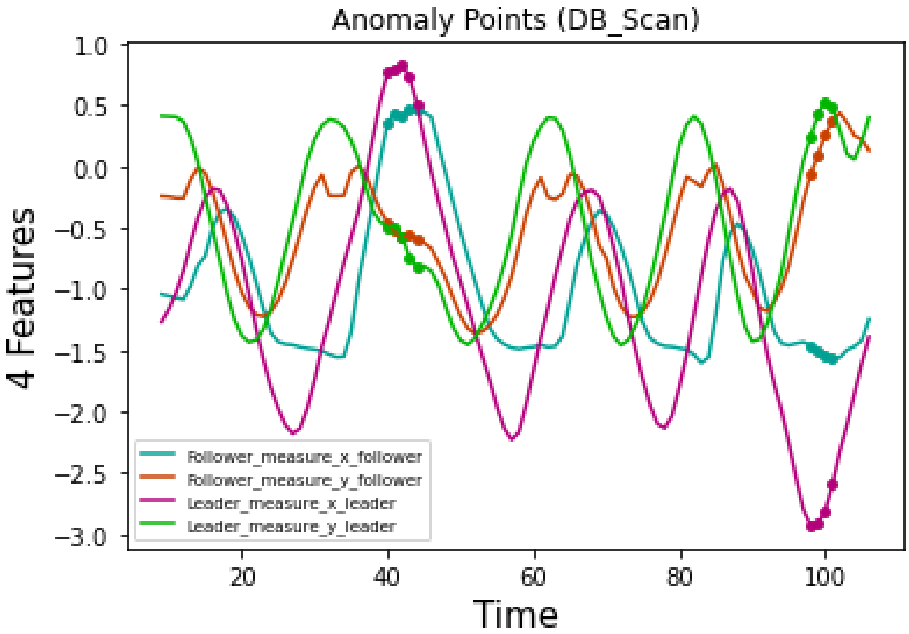
- **hyperparameter tuning based on the highest accuracy**

We found that best values for epsilon is 0.7, and for min_samples is 13 based on the highest accuracy.

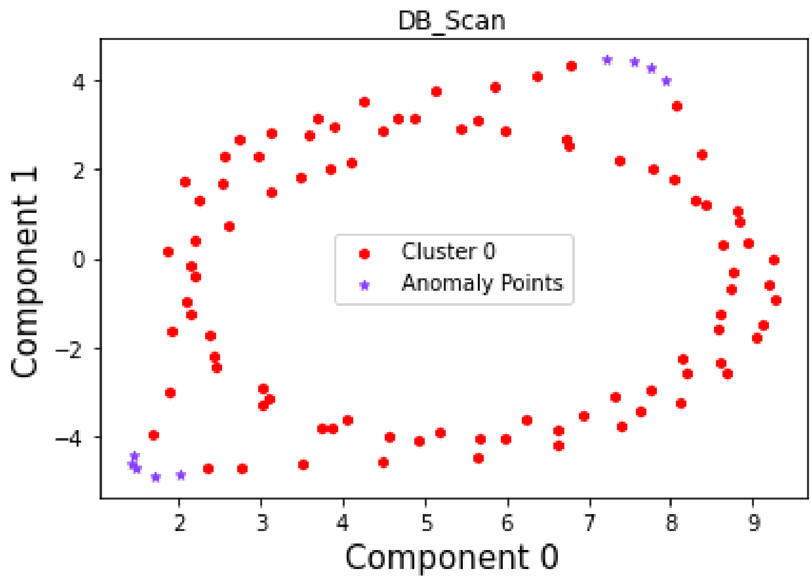
```
# After hyperparameter tuning we found that best values for epsilon is 0.7, and for for min_samples is 13
#------Train the model and make predictions------
model = DBSCAN(eps=0.7, min_samples=13)
DBscan_Clusters = model.fit_predict(X)
DBscan_Clusters = pd.DataFrame(DBscan_Clusters).replace(-1,1)

# Confusion Matrix and Report

DB_Report = classification_report(Y, DBscan_Clusters)
DB_Cf = ConfusionMatrix(Y, DBscan_Clusters)
PLOT_ConfusionMatrix(DB_Cf,'DB_Scan')

#---------------------DB_Scan Time Series (Anomaly Points)---------------------
TimeSeries_plt(TimeSeries)
PltDetected_Points(TimeSeries, DBscan_Clusters,'Anomaly Points (DB_Scan)')

#---------------------Tsne DBSCAN---------------------
XTsne = T_SNE(X)
XTsne = pd.concat([pd.DataFrame(XTsne), pd.DataFrame(DBscan_Clusters)],axis=1 , ignore_index = True).astype(float)
DB_Ls = GetListOfClasses(2, XTsne,  pd.DataFrame(XTsne).columns[2])
PlotDataPoints(2, DB_Ls, 'Component 0','Component 1' ,'Cluster 0','Anomaly Points' ,20 , 'DB_Scan').show()
```



- **DB_SCAN Confusion matrix**



- **Time Series with Anomaly Points for DB_SCAN**



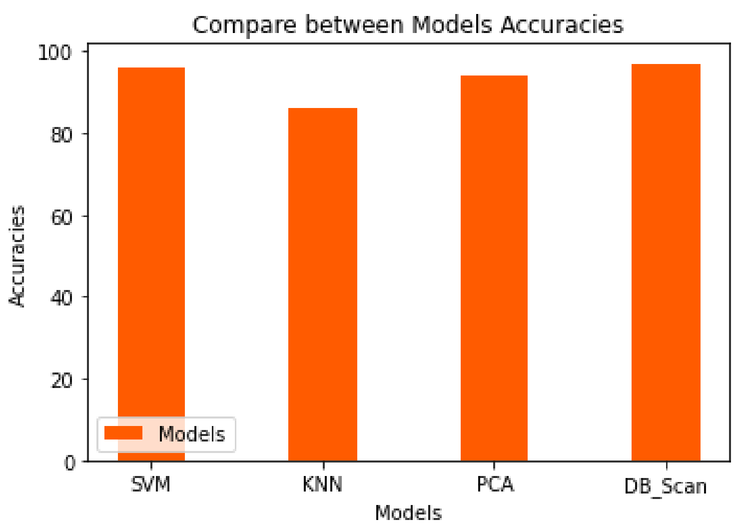
- **T-SNE for DB_SCAN**

	precision	recall	f1-score	support
0.0	0.97	1.00	0.98	86
1.0	1.00	0.75	0.86	12
accuracy			0.97	98
macro avg	0.98	0.88	0.92	98
weighted avg	0.97	0.97	0.97	98

- **Classification Report for DB_SCAN**

9. Compare between Models Accuracies

```
#--------------------Compare between Models Accuracies--------------------
Bar(['SVM','KNN','PCA','DB_Scan'], [96,86,94,97], 0.4, '#FF5B00' , 'Models', 'Accuracies' , 'Compare between Models Accuracies', 'Models')
```



- **Bar Chart to Compare between Models Accuracies**

10. Conclusion

- After we detected anomalies using four different models SVM, PCA, KNN and DBSCAN we have plotted 2D TSNE and Time series to visualize the movement of Qbot and Qdrone and to see if they are moving in a correct way or not, after we applied confusion matrix and report classification for every model and compare between them, DBSCAN model gave us the highest accuracy, and also the highest precision and the highest f1 score (for both anomaly and normal instances).

DB_SCAN: -

	precision	recall	f1-score	support
0.0	0.97	1.00	0.98	86
1.0	1.00	0.75	0.86	12
accuracy			0.97	98
macro avg	0.98	0.88	0.92	98
weighted avg	0.97	0.97	0.97	98

SVM: -

	precision	recall	f1-score	support
0.0	0.97	0.99	0.98	86
1.0	0.90	0.75	0.82	12
accuracy			0.96	98
macro avg	0.93	0.87	0.90	98
weighted avg	0.96	0.96	0.96	98

PCA: -

	precision	recall	f1-score	support
0.0	0.95	0.98	0.97	86
1.0	0.80	0.67	0.73	12
accuracy			0.94	98
macro avg	0.88	0.82	0.85	98
weighted avg	0.94	0.94	0.94	98

KNN: -

	precision	recall	f1-score	support
0.0	0.88	0.97	0.92	86
1.0	0.25	0.08	0.12	12
accuracy			0.86	98
macro avg	0.57	0.52	0.52	98
weighted avg	0.81	0.86	0.82	98