Computer Engineering Department National University of Technology Islamabad, Pakistan

Introduction to Data Mining

Practice Exercise 04

Name:	Muhammad Sami Uddin Rafay
Roll Number:	F18604013
Submitted To:	Dr. Kamran Javed
Date:	30 November 2020

Practice Exercise 04

Feature Extraction

Objective:

- Determining appropriate class for new data points.

Equipment/Software Required:

- Python (Spyder 4.0 Anaconda Distribution)

Background:

Assume that we have a learning/ training data set of a machine which indicates healthy and faulty states of measurements (data points) from hour 1 to H (H=20). Given threshold is 0.5. We want to determine state of a new machine by comparing its measurements with learned data:

Tasks:

1. Create training data set (20rxc3):
   a. Hour Health State (0= healthy, 1= faulty), note: health feature has random values from 0-1.
   b. Show a scatter plot of hour vs health features.
2. Initially the labels are fixed your task is to set state (class labels) according to the given threshold value. i.e., if health measurement value is >=0.5 state 1 otherwise 0.
3. Get the input from user that should be <=1 (i.e., new observation newpt)
4. Compute the distance of newpt with each observation of health feature, i.e., vector D. Plot distance feature D.
5. Use find command to determine indexes of distance D vector value <= threshold and the corresponding states on the same indexes from state feature.
6. Use mode command to determine most similar class and display state of new data point i.e. health or faulty.
7. Show a scatter plot of hour vs sorted health features and based on the threshold fill the healthy states blue and faulty states red. Also plot the new observation on the scatter plot (note the time index can be of your choice, i.e. hour 5 may be hour).
8. Plot states of the healthy and faulty class data.

Code:

```
# Feature extraction from Time Series Data

# importing neccessary python pakages

import matplotlib.pyplot as plt
import numpy as np
import random
import statistics
import pandas as pd


# defining function for generating random temperature values
```

```
def random_floats(low, high, size):
    return [random.uniform(low, high) for _ in range(size)]
print('\n')


# defining array for storing respective values

Temperature_yearly=np.array([])
Temperature=np.array([])
Temperature_mean=np.array([])
Temperature_var=np.array([])


# generating Time Series Temperature, Mean per day, Variance per day
for i in range(365):
    Temperature=random_floats(0, 50, 24)
    Temperature_yearly=np.append(Temperature_yearly,Temperature)
    Temperature_mean=np.append(Temperature_mean ,np.mean(Temperature))
    Temperature_var=np.append(Temperature_var, statistics.variance(Temperature))


# printing shape of respective arrays to confirm the total values
print(Temperature_yearly.shape)
print(Temperature_mean.shape)
print(Temperature_var.shape)


# intializing figure 1 for plotting Temperature values

plt.figure(1, figsize=(10,8))


# plotting temperature per year

plt.subplot(111)
plt.plot(Temperature_yearly, color='purple')
plt.xlabel("Time (One Year)")
plt.ylabel("Temperature")
plt.grid()


# intializing figure 2 for plotting Mean Temperature values

plt.figure(2, figsize=(8,6))


# plotting Mean temperature per day

plt.subplot(111)
plt.plot(Temperature_mean, color='brown')
plt.xlabel("Time (365 Days)")
plt.ylabel("Mean(average) Temperature per Day")
plt.grid()
```

```
# intializing figure 2 for plotting Mean Temperature values

plt.figure(3, figsize=(6,4))

# intializing figure 3 for plotting Variance Temperature values

plt.subplot(111)
plt.plot(Temperature_var, color='gray')
plt.xlabel("Time (365 Days)")
plt.ylabel("Variance Temperature per Day")
plt.grid()

# showing all graphs

plt.show()


# Selecting any random value of variances as threshold

threshold=random.choice(Temperature_var)
print("\n")
print("Threshold : ")
print("\n")
print(threshold)


# defining an array for saving variance values greater than threshold

days_above_threshold=np.array([])


# designing algorithm to sort out the day having variance values greater than Threshold

for i in Temperature_var:
        days_above_threshold=np.append(days_above_threshold,np.where(Temperature_var>threshold))

# printing variances above threshold
print("\n","Days above Threshold : ","\n")
print(days_above_threshold)



# converting days_above_threshold in DataFrame

day_above_DataFrame=pd.DataFrame(days_above_threshold)


# exporting days_above_threshold in .csv

day_above_csv=day_above_DataFrame.to_csv("D:\day_above_DataFrame.csv")
```

Output:

```
Please Enter a number less than 1 :0.6
```

Please Enter the index for new Entry :13

Health Condition :

```
[0.8957469950770193, 0.08077845796976235, 0.1783230740000642, 0.8196972267071088,
0.034014818235133415, 0.8065926392152845, 0.3065121095402894, 0.6145450614235138,
0.6604469817227682, 0.937840482168668, 0.8075594592804584, 0.15166735485838756,
0.0020670767195170026, 0.7506835619814233, 0.06885024904216963, 0.6921701015609686,
0.15236045131613207, 0.2281242334740049, 0.38522690404007975, 0.6301966132141353]
```

After Discretization :
Good : 0
Faulty : 1

health_condition :

```
[1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
```

Distance Vector :

```
[0.295747   0.51922154 0.42167693 0.21969723 0.56598518 0.20659264
 0.29348789 0.01454506 0.06044698 0.33784048 0.20755946 0.44833265
 0.59793292 0.15068356 0.53114975 0.0921701  0.44763955 0.37187577
 0.2147731  0.03019661]
```

Indexes at which D vector <= threshold

```
[ 1.  2.  4.  6. 11. 12. 14. 16. 17. 18.]
```

Corresponding states on the same indexes from state feature :

```
[0.08077846 0.17832307 0.03401482 0.30651211 0.15166735 0.00206708
 0.06885025 0.15236045 0.22812423 0.3852269 ]
```

The Most Similar class :

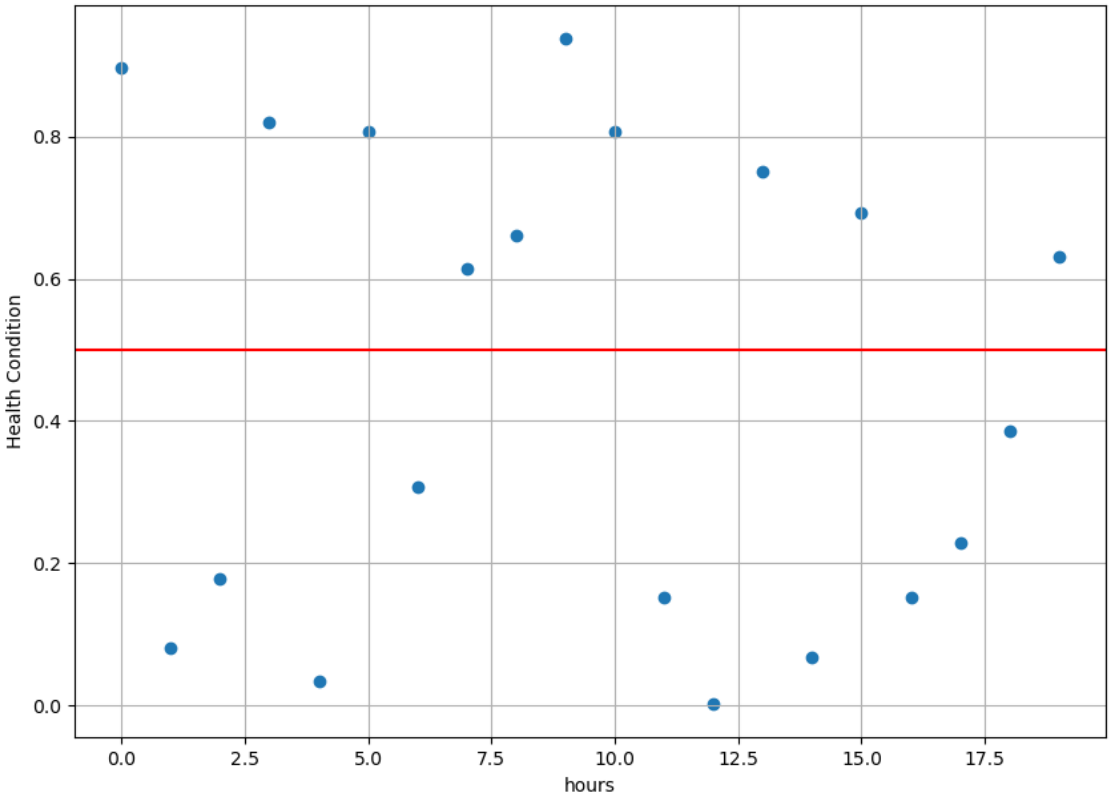
```
[(1.0, 10)]
```

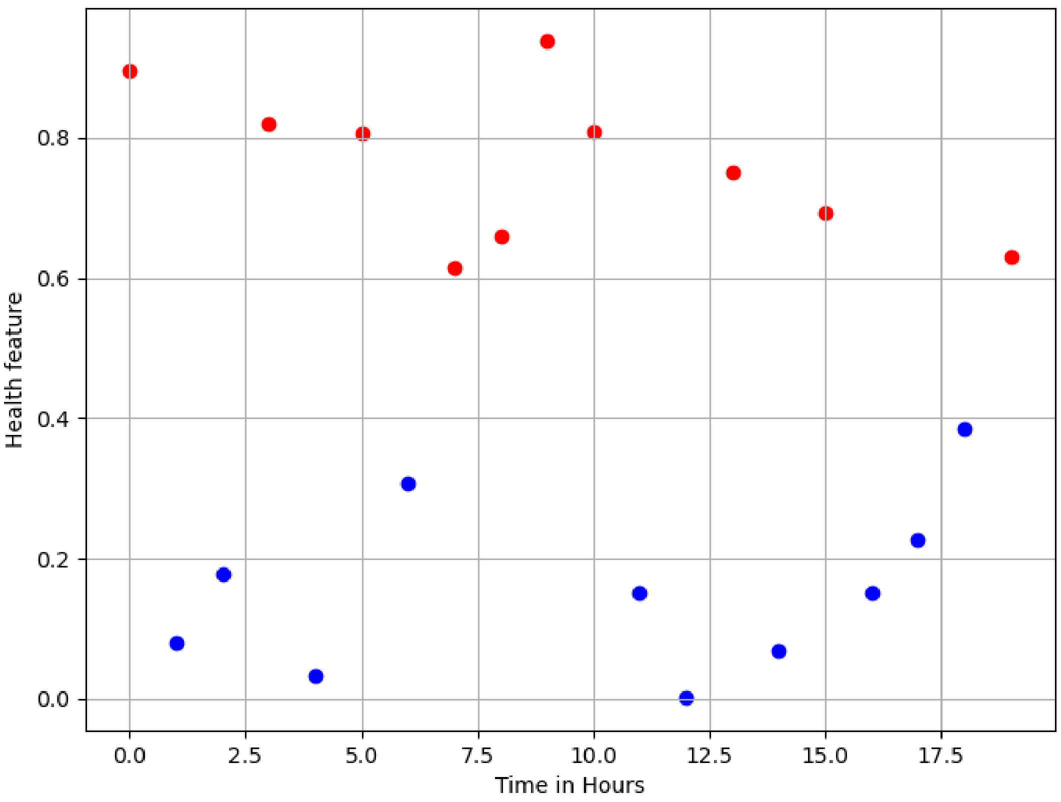
New Entry is Faulty

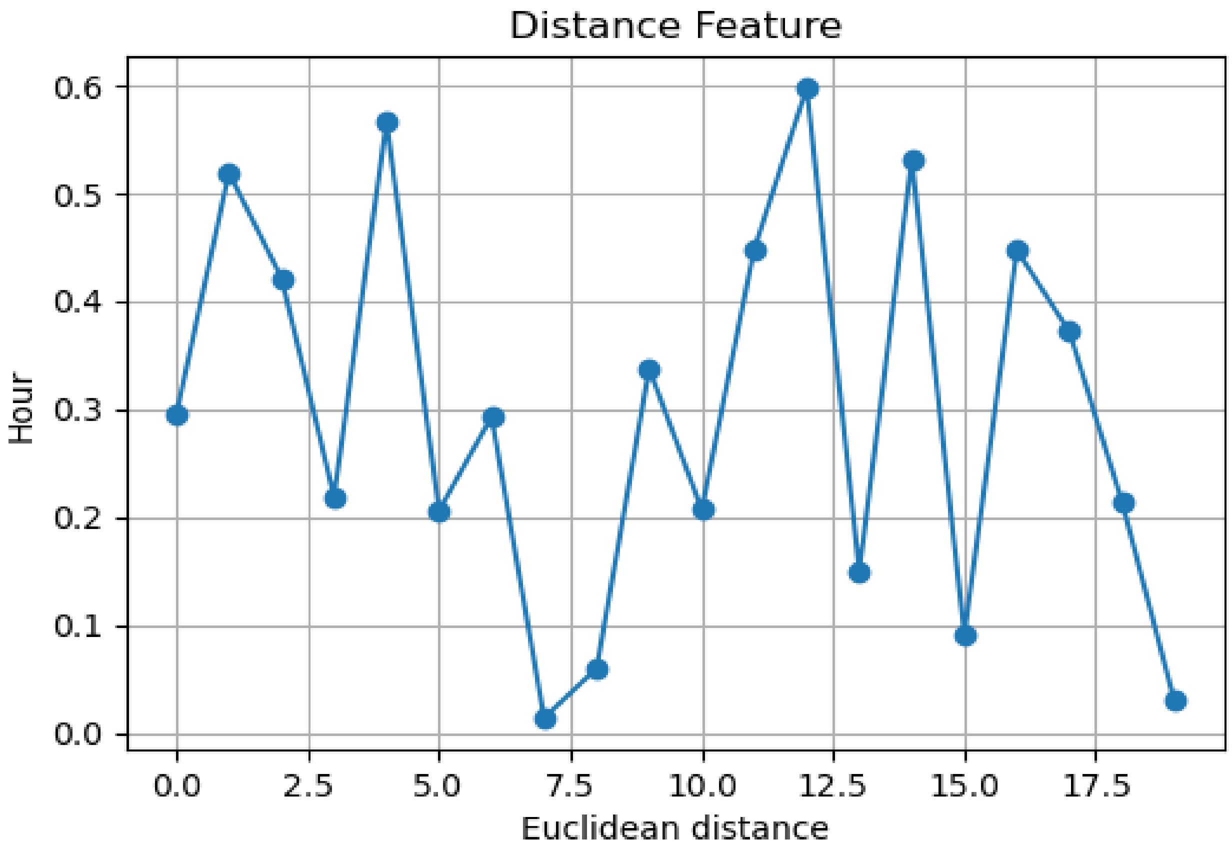
Sorted Health Features :

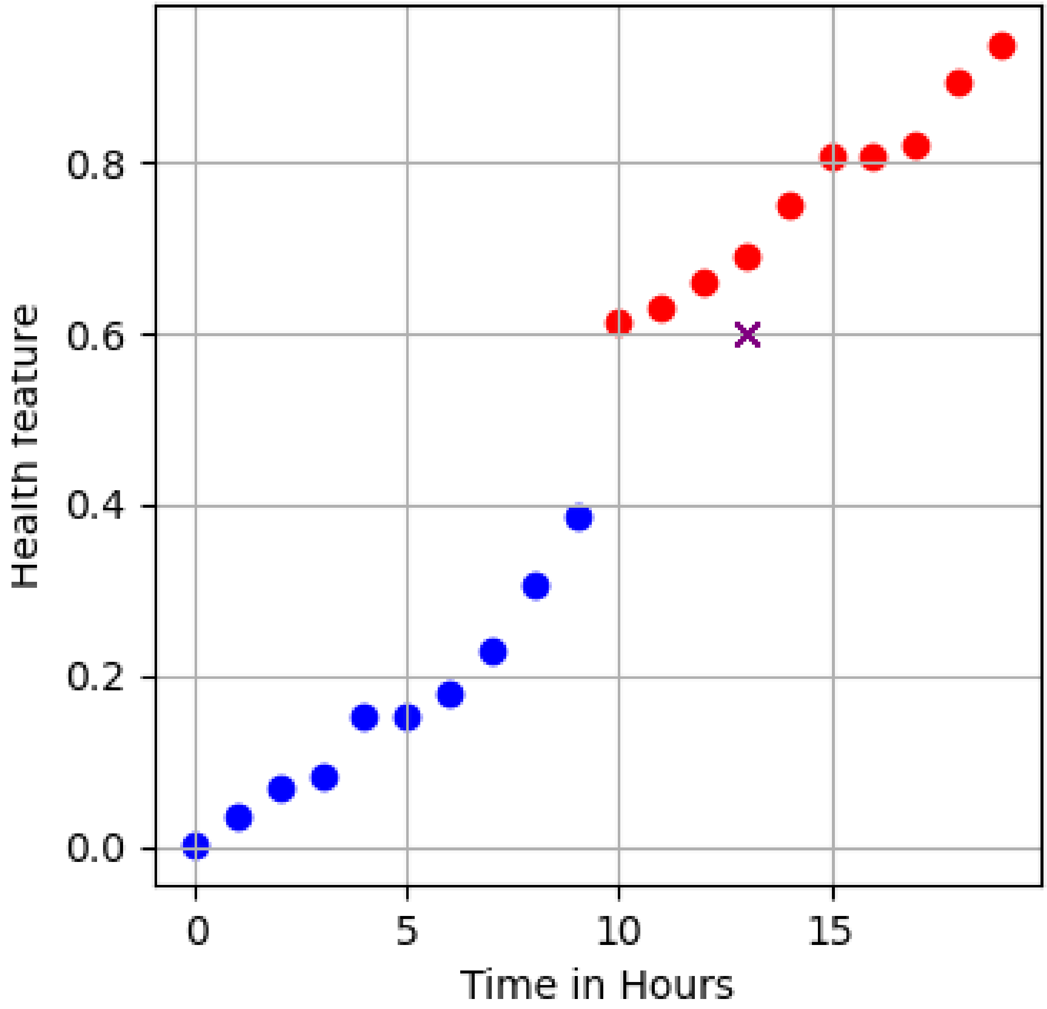
[0.0020670767195170026, 0.034014818235133415, 0.06885024904216963, 0.08077845796976235, 0.15166735485838756, 0.15236045131613207, 0.1783230740000642, 0.2281242334740049, 0.3065121095402894, 0.38522690404007975, 0.6145450614235138, 0.6301966132141353, 0.6604469817227682, 0.6921701015609686, 0.7506835619814233, 0.8065926392152845, 0.8075594592804584, 0.8196972267071088, 0.8957469950770193, 0.937840482168668]

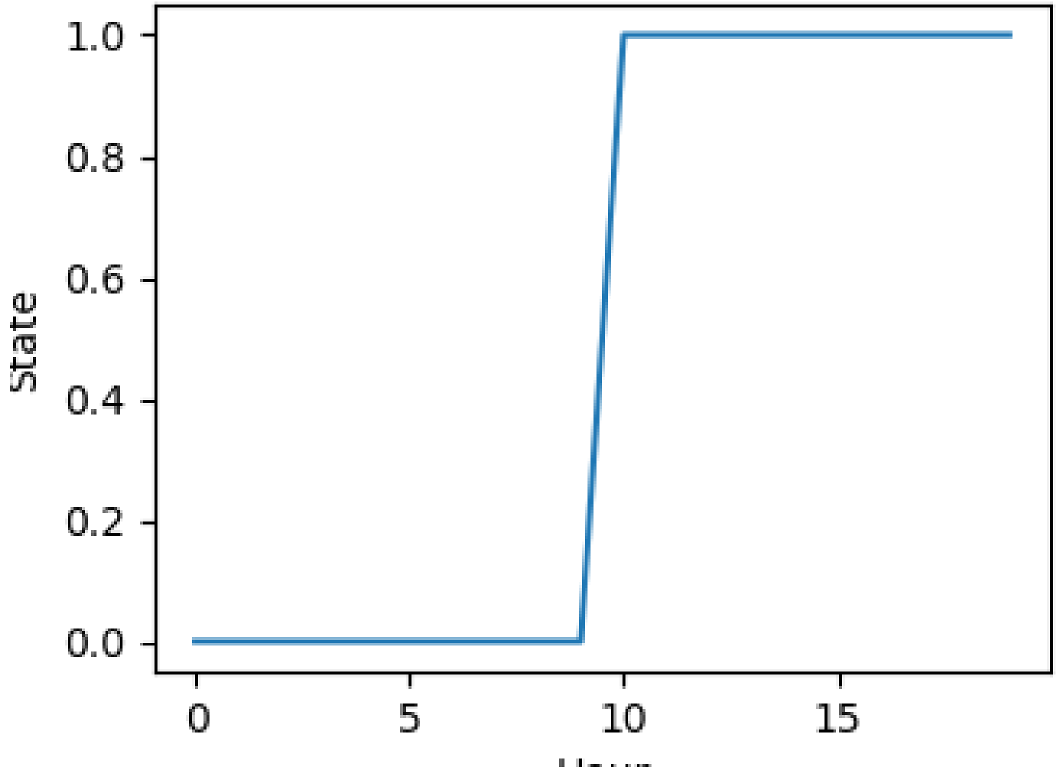
Graphs:




Health feature
0.0
0.2
0.4
0.6
0.8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Time in Hours


Distance Feature
Hour
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Euclidean distance


0.8
0.6
0.4
0.2
0.0
Health feature
0
5
10
15
Time in Hours


1.0
0.8
0.6
0.4
0.2
0.0
State
0
5
10
15
Hour

Results and Discussions:

In this practical I learned to discretize the health condition numerical data facts into binary form (0 and 1/ Good and Faulty) and try to extract the features from it. Also trained a model to estimate the new candidate health condition and its Euclidean distance from all data entries in the dataset. The results are changing at each run because of random data selection. And most especially the scatter plot helped a lot to do this in efficient way.

The python packages I used in this practical: -

- ✓ NumPy
- ✓ matplotlib
- ✓ pandas
- ✓ random

Conclusion:

What I found about Feature Extraction is used when we need to reduce dimensions of a dataset without losing important and respective information.

.py file is attached.