IoT exercises – Week 11-12

For the final 2 weeks, we will review what we have learned and practiced. We will first briefly look at the AI in IoT, specifically some naive patter recognition algorithms like K-Means and KNN for implementation (*Please note that for simplicity, we will get the benchmark dataset and bespoke modules in Python for you to practice. The key idea is to know how to implement the algorithms step by step). We will further look at some modules of NodeMCU/ESP8266 that have not been covered yet. And as a summary, we will modularize the functions we have implemented. A smart LED lantern supported by NodeMCU/ESP8266, multiple sensors and mobile app based remote control will be created by yourself. Please feel free to explore more possible applications and don't limit yourself to the practical materials.

Should you need any other resources, please let me know it.

The official documentation is provided here again for your reference. Please remember, when you are building your own IoT project in the future, always refer to the bespoke built-in modules and read their documentation first, which will be helpful!!!

https://nodemcu.readthedocs.io/en/master/

For exercises 1-5, you are expected to use the Python environment for programming. However, it is also flexible to use a different language and libraries if you are more familiar with other programming languages.

Exercise 1:

In this exercise, you will need to load the Iris Dataset and Handwritten Digits Dataset using pandas, sklearn, matplotlib. Read the description of the dataset, know the features, the target values and their dimensionality. Particularly, check what the target values look like. Please (always remember to) refer to official documentation for lib and function details in you future application. Here we refer to the sklearn and matplotlib libs in Python.

sklearn (https://scikit-learn.org/stable/)

matplotlib (https://matplotlib.org/)

==

```
iris = datasets.load_iris()
```

```
print(iris)

irisx = iris.data

print(irisx.shape)

irisy = iris.target

print(irisx.shape)


digits = datasets.load_digits()

print(digits)

digitsx = digits.data

print(digitsx.shape)

#1797 samples/records

#each ssample/record is an 8x8 image of digits

#the 8x8 matrix is stored as a vector

digitsy = digits.target

print(digitsy.shape)

#the label of each sample/record

plt.gray()

plt.matshow(digits.images[990])

#The images are kept

plt.show()
```

==

Now you have loaded the two datasets and have a brief understanding of the features and the target values. In this exercise, you will need to implement the simple tool of confusion matrix to provide a more intuitive understanding of your predicted labels in comparison to the true labels. Libs of sklearn, matplotlib and seaborn will be used. Or you can update your sklearn lib to the latest version to directly use plot_confusion_matrix function to display the confusion matrix.

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

seaborn (https://seaborn.pydata.org/)

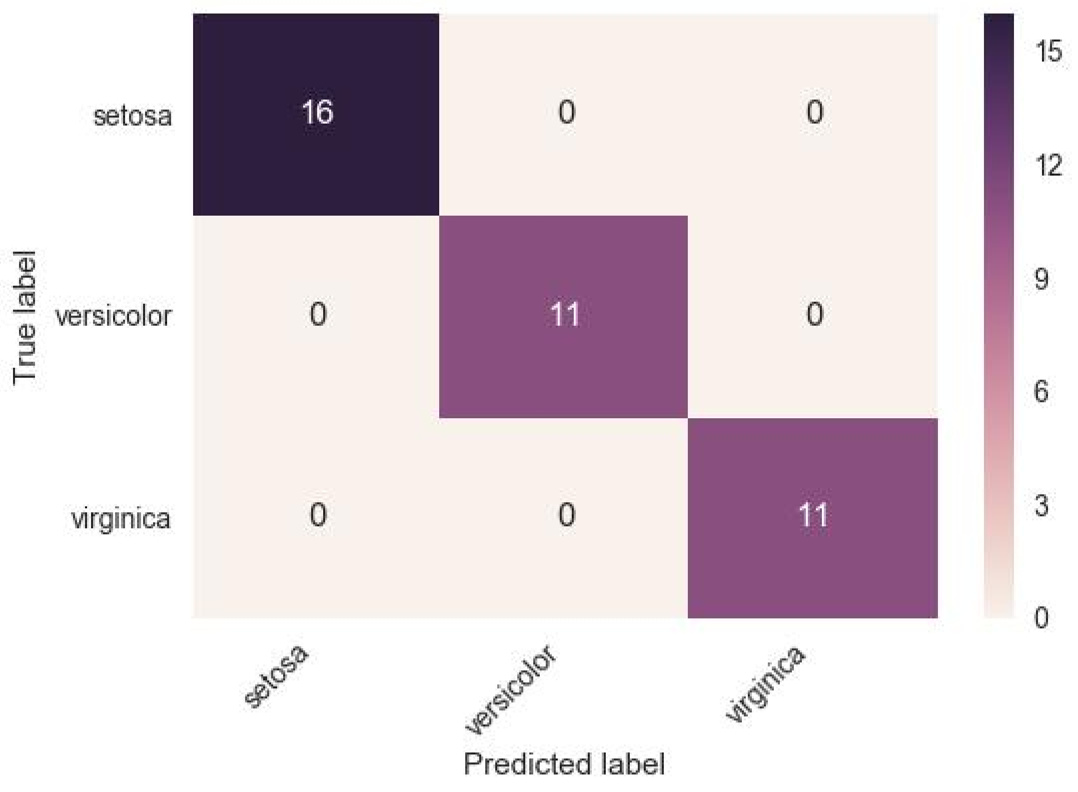
==

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix

#from sklearn.metrics import plot_confusion_matrix

#you can also update your sklearn lib to the latest version to use

#plot_confusion_matrix

#https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn import datasets

from sklearn.model_selection import train_test_split


def confusionM(y_true,y_predict,target_names):

#function for visualisation

    cMatrix = confusion_matrix(y_true,y_predict)

    df_cm = pd.DataFrame(cMatrix,index=target_names,columns=target_names)

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

    cm = sns.heatmap(df_cm,annot=True,fmt="d")

    cm.yaxis.set_ticklabels(cm.yaxis.get_ticklabels(),rotation=90)

    cm.xaxis.set_ticklabels(cm.xaxis.get_ticklabels(),rotation=0)

    plt.ylabel('True label')

    plt.xlabel('Predicted label')


iris = datasets.load_iris()

X = iris.data

y = iris.target

target_names = iris.target_names

X_train, X_test, y_train, y_true = train_test_split(X, y)
```

```
lda = LinearDiscriminantAnalysis()

lda.fit(X_train,y_train)

y_predict = lda.predict(X_test)

confusionM(y_true,y_predict,target_names)
```

==

A confusion matrix should look like



Exercise 2:

Now we go on with the application of the clustering algorithm of K-Means using bespoke modules from sklearn on the Iris Dataset and Handwritten Digits Dataset respectively, and ouput the confusion matrix based on Exercise 1 for your prediction results.

The key steps and some bespoke functions you might use for the implementation of clustering for label prediction are given as follows.

- **Split the data into training set and testing set**

```
from sklearn.model_selection import train_test_split
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

- **Train the model with the training set**
- **Test the model with the testing set**
- **Predict the target value and compare to the true value**

```
from sklearn.cluster import KMeans
```

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

- **Output the error rate and confusion matrix**
- **Change the parameter initialization and check how the parameter settings have affected your results**

(Optional) Exercise 3:

Now you have successfully used the bespoke modules for clustering in Python (or the programming language that you chose). Please implement the K-Means clustering model by coding each step by yourself. And then compare your implementation with the module from sklearn to see if there exists any difference between efficiencies. The Big-O analysis is recommended.

Steps for K-Means:

1. Initialise the K and K random initialised centroids
2. Repeat the following until convergence (cluster centroids don't change) or reach the predefined maximum number of iterations
   a. Calculate the distance from each sample to each centroid
   b. Assign the sample to the cluster (shortest distance to its centroid)
   c. Calculate the mean value of all samples within each cluster
   d. Update the centroids with the mean value

Exercise 4:

The Iris Dataset and Handwritten Digits Dataset are naturally suitable for classification methods. Particularly in this exercise, we will apply the simplest classification algorithm K-Nearest Neighbours with a varying parameter K for practice. Note that because of the nature of dataset with labels for a classification problem, we don't need to convert the clusters to the standard labels.

Please (always remember to) refer to official documentation for lib and function details.

sklearn (https://scikit-learn.org/stable/)

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

==

```
import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from sklearn.metrics import confusion_matrix

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

def confusionM(y_true,y_predict,target_names):

#function for visualisation

    cMatrix = confusion_matrix(y_true,y_predict)

    df_cm = pd.DataFrame(cMatrix,index=target_names,columns=target_names)

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

    cm = sns.heatmap(df_cm,annot=True,fmt="d")

    cm.yaxis.set_ticklabels(cm.yaxis.get_ticklabels(),rotation=90)

    cm.xaxis.set_ticklabels(cm.xaxis.get_ticklabels(),rotation=0)

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

iris = datasets.load_iris()

X = iris.data

y = iris.target

target_names = iris.target_names

X_train, X_test, y_train, y_true = train_test_split(X, y)

nn = KNeighborsClassifier(n_neighbors=1)

nn = KNeighborsClassifier(n_neighbors=3)

nn = KNeighborsClassifier(n_neighbors=5)

nn = KNeighborsClassifier(n_neighbors=10)

#from NN to 3-NN to 5-NN to 10-NN
```

```
nn.fit(X_train,y_train)

y_predict = nn.predict(X_test)

print(y_predict)

confusionM(y_true,y_predict,target_names)
```

==

(Optional) Exercise 5:

Following the Exercise 4, this time you are asked to implement the K-Nearest Neighbours step by step by yourself. Please make sure that you can implement an algorithm with the description/steps given.

HINT: Decompose the algorithms into smaller parts and define the input and output. Then Implement each part individually and integrate them all.

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Exercise 6:

In this exercise, you will need to practice on how to modularise, or we say encapsulation in programming, the functions in NodeMCU/ESP8266. Modularisation allows you to divide functions into multiple files.

==

```
module = { }

module.value = 1

function   module.func()

   print(module.value)

end

return module
```

==

When you are using the module, you simply call require "module" in the lua file, "module" is the name of the module you want to use.

Now we further expand the sample module into your GPIO based LED control module.

==

```
myIO = { }

local led1, led2 = 4, 0

local led1mode, led2mode = gpio.OUTPUT, gpio.OUTPUT

function myIO.gpioInit()

    gpio.mode(led1, led1mode)

    gpio.mode(led2, led2mode)

    return true

End

function myIO.setLED(x)

    if x == 1 then

        gpio.write(led1, gpio.LOW)

        gpio.write(led2, gpio.HIGH)

    else

        gpio.write(led2, gpio.LOW)

        gpio.write(led1, gpio.HIGH)

    end

End

return myIO
```

==

Now you can test the module with the following 4 lines in your main Lua file.

```
local myIO = require "myIO"

myIO.gpioInit()

myIO.setLED(1)

myIO.setLED(0)
```

Please note that the current "myIO"module is not fully complete. Consider how to add any control command to this module. And also consider how you could modularise the functions that you have implemented in previous weeks into individual modules.

(Optional) Exercise 7:

In this exercise you are asked to develop a mobile application for the human-machine interaction between yourself and the IoT device. If you don't have any previous experience in mobile app development, a simple solution is to use the bespoke creators like App Inventor (https://appinventor.mit.edu). You need to design the interactions with logics and variables with the simple GUI. The design can be flexible according to your need, which can also be included in your coursework.

Exercise 8:

There are some bespoke modules not used yet. Here only a quick review is given to you, if you are interested, to practice with.

uart module: The universal asynchronous receiver-transmitter (UART) module is available in all microcontroller and embedded systems. It is used for testing of receiving and transmitting data between devices/sensors. Most modern systems are provided with powerful IDE and testing environments, the UART functions are not commonly used. https://nodemcu.readthedocs.io/en/master/modules/uart/

node module: This provides functions on the system level. It can be used to read info about the systems and chips like chip ID, flash ID, flash size, firmware version, working mode, and system frequency, etc. And some system-level parameters can be set like the CPU frequency (only binary choice of 80MHz or 160MHz), working mode of the NodeMCU/ESP8266 (awake or sleeping mode), etc. There are also functions like rebooting the system, compiling the Lua file into bytecode, and restoring factory defaults, etc. https://nodemcu.readthedocs.io/en/master/modules/node/

Please refer to the official documentation to explore more by yourself.

Now you have done all the exercises of the IoT unit. Please check if you are able to:

- Know sensory principles, communication protocols and algorithms applied in IoT.
- Analyse key steps and basic operations, then provide a plan when given an IoT problem.
- Design a low-cost and simple platform of IoT with bespoke sensors, controllers and modules.

Extra for you to practice:

You are encouraged to reform some of your own small device ("small" means they are powered by a supply below 5V) currently without an access to the Internet, get it online and make it remotely controllable and readable. (this is totally optional and not included in the final portfolio).

Please always stay careful about the safety concerns. **STAY SAFE** when you are reforming your own device.

Thank you for your support to this module!

I do wish you all the best in your future career!