Machine Learning for IoT
Lab 5 – REST

HowTo 1: JSON in Python

The Python *json* package enables to convert Python dictionaries to JSON (and vice versa).
`json.loads(json_string)`: convert a JSON string to a Python object

```
string = '{"name": "Tony", "surname": "Stark"}'
obj = json.loads(string)
```

`json.dumps(object)`: convert a Python object to a JSON string

```
obj = {"num1": 12,"num2": 34}
string = json.dumps(obj)
```

Exercise 2: RESTful Calculator (two operands)

Design a RESTful **web service** that implements a simple calculator. Develop the **HTTP GET** method to manage the following commands:

- **add**: to add two operands and send in the *HTTP body* the JSON.
- **sub**: to subtract two operands and send in the *HTTP body* the JSON.
- **mul**: to multiply two operands and send in the *HTTP body* the JSON.
- **div**: to divide two operands and send in the *HTTP body* the JSON. Check that the operation is possible. If not, an exception must be raised with the suitable HTTP code.

The output should be a JSON reporting both input operands, the executed command, and the result. **Manage possible errors in invoking the web services (e.g., wrong command or wrong number of parameters).**

EXAMPLE*:*
`http://localhost:8080/add?op1=10&op2=12.2`

OUTPUT JSON:
{
"command": "add",
"op1": 10.0
"op2": 12.2
"result": 22.2
}

where **add** is the command and the parameters **op1** and **op2** provide the input operands.

SUGGESTION: Use **Postman** to test the RESTful web services and validate the JSON with http://jsonlint.com.

Exercise 3: RESTful Calculator (multiple operands)

Extend Ex. 2 to design a RESTful-style calculator that process a list of numerical values, instead of two. Develop the **HTTP PUT** method for receiving the following JSON in the body-message:

{
"command": "add",
"operands": [10, 9, 8, 7, 6, 5, 3, 2, 1]
}

where *"command"* indicates the operation to be performed among **add**, **sub**, **div**, **mul**. *"operands"* is an array with the inputs for the operation. Finally, the **PUT** method should return a JSON reporting the input operands, the executed command, and the result.

Exercise 4: RESTful Calculator Client

4.1. Develop a **client** application to invoke the RESTful calculator developed in Ex. 2 that:

- asks to end-users the operation to be performed and the two operands.
- invokes the RESTful web service.
- shows the results on the shell to end-users in a friendly way (NOT the full JSON).

SUGGESTION: use the *requests* Python package to send HTTP requests and read their contents (take care in handling exceptions).

SYNOPSIS:
```
python twooperands_client.py add 10 12.2
```

OUTPUT:
```
10.0 + 12.2 = 22.2
```

4.2. Develop a **client** application to invoke the RESTful calculator developed in Ex. 3.

SYNOPSIS:
```
python multipleoperands_client.py add 10 9 8 7
```

OUTPUT:
```
10.0 + 9.0 + 8.0 + 7.0 = 34.0
```

Exercise 5: RESTful Data Collection

5.1. On the Raspberry Pi, develop a RESTful **web service** and identify the proper HTTP methods (among GET, POST, PUT and DELETE) to:

- Retrieve information about the temperature sensor
- Retrieve information about the humidity sensor
- Retrieve a 1s audio signal recorded with the USB microphone
- (optional) Retrieve an image captured with the PiCamera

5.2. Use the SenML+JSON data format to exchange the information among the clients.
For the audio signal and the image use the attribute "vd" defined in SenML to send data and base64 strings as arrays (use the *b64encode* method of the *base64* package to get base64 bytes and the *decode* method to get the base64 string).

5.3. On your notebook, develop a **client** application to retrieve such information every 30 seconds. Use the library *requests* to open URLs and read their contents (handle exceptions properly).

- Print the timestamp (in human-friendly format), temperature and humidity on the shell
- Store the audio signal on disk in *.wav* format, using the POSIX timestamp as filename.
- (optional) Store the image in *.png* format, using the POSIX timestamp as filename.

SYNOPSIS:
```
python collector_client.py
```

OUTPUT (on screen):
```
10/12/2020 20:46:24 19Cel 24%RH
10/12/2020 20:46:56 19Cel 24%RH
```

OUTPUT (files):
```
1607629584.wav 1607629616.wav
1607629584.png 1607629616.png
```

Exercise 6: RESTful Inference for Keyword Spotting

the reasmpling should be done on the board not notebook

6.1. On your notebook, develop a RESTful **web service** that receives a one-second audio signal (resampled at 16 kHz) in SenML+JSON format and classifies the keyword contained in the signal. The web service should support different prediction models. The model name is given in the URL (possible options are mlp, cnn, dscnn).
The output is a JSON containing the label and the probability score.

EXAMPLE:
```
http://localhost:8080/mlp
```

OUTPUT JSON:
{
"label": "yes"
"probability": 99.31
}

6.2. On your Raspberry PI, develop a **client** application that records a one-second signal, sends the data to the web service, and prints the received prediction on the screen in human-friendly format.

OUTPUT:
```
yes (99.31%)
```