AU 332 Artificial Intelligence: Principles and Techniques

By: ZheHao Huang (518021910660)

HW#: 0

September 10, 2020

I. INTRODUCTION

A. Purpose

The goal of today's lab is to get familiar with Python and Latex. And based on the homework template, to write my first test report with Latex. Python is one of the most popular programming languages for data science. The lab guides me through basics of Python for the Deep Learning course and provides some useful references.

In this lab, we will first follow the instruction to setup a scientific Python environment with Anaconda which helps us manage our test environments and various Python packages. Second we follow the tutorial to run some examples in the Jupyter Notebook and have a grasp of basic Python and some useful packages such as Numpy, SciPy and Matplotlib. Finally we're required to finish three Python programming questions and write a test report with Latex.

B. Equipment

This lab is mainly programming tutorial which does not need other hardware equipment but a computer:

- Computer capable of running the software mentioned

C. Procedure

1. Learn from the Python tutorial.
2. Set up the scientific Python environment.
3. Solve three Python programming questions.
4. Learn from the Latex tutorial.
5. Write the test report based on the homework template with Latex.
6. Submit the test report and the source code.

II. QUESTIONS TO SOLVE

A. Addition

Instruction: Open `addition.py` and look at the definition of `add`:

```
def add(a, b):
    # Return the sum of a and b
    # *** YOUR CODE HERE***
    return 0
```

Please modify this definition to return the sum of a and b.

Method: We just need to use `+` to add a and b to get the result, I not only return the sum of a and b, but also print the result.

Source Code

```
# addition.py


def add(a, b):
    # print the sum of a and b
    print("a + b = ", a + b)
    # return the sum of a and b
    return a + b
```

B. buyLotsOfFruit function

Instruction: Add a `buyLotsOfFruit(orderList)` function to `buyLotsOfFruit.py` which takes a list of `(fruit,pound)` tuples and returns the cost of your list. If there is some `fruit` in the list which doesn't appear in `fruitPrices` it should print an error message and return `None`. Please do not change the `fruitPrices` variable.

Method: We are provided with `fruitPrices` like follows

```
fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75, 'limes': 0.75, 'strawberries': 1.00}
```

I first create a variable `totalCost` to store the cost of the orderList. And then use `for` loop over the input list while in each iteration I use `if...in...` sentence to check if the fruit is in the `fruitPrices`. If that's true, I multiply the price of that fruit by its required pound and add the result to the `totalCost`. But if that's false, I'll print a error message and directly return `None` without continuing the loop.

Something we need to consider is that the input list can be empty and in this case we will just return the `totalCost` of 0.0 since there is no fruit we need to take into account.

Algorithm 1 buyLotsOfFruit

Input: An list of (fruit,pound) tuples, orderList.
Output: The total cost of the list, totalCost; or None;

$totalCost \leftarrow 0$
for each $fruit, pound \in orderList$ **do**
  if $fruit \in fruitPrices$ **then**
    $totalCost \leftarrow totlaCost + fruitPrices[fruit] * pound$
  else
    print ERROR message
    return *None*
  end if
end for
return *totalCost*

Source Code

```python
# buyLotsOfFruit.py
"""
To run this script, type

  python buyLotsOfFruit.py

Once you have correctly implemented the buyLotsOfFruit function,
the script should produce the output:

Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
"""

fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75,
               'limes': 0.75, 'strawberries': 1.00}


def buyLotsOfFruit(orderList):
    """
        orderList: List of (fruit, numPounds) tuples

    Returns cost of order
    """
    totalCost = 0.0
    for (fruit, numPounds) in orderList:
        if fruit in fruitPrices:
            totalCost += fruitPrices[fruit] * numPounds
        else:
            print("ERROR: %s does not appear in fruitPrices!" % fruit)
            return None
    return totalCost


# Main Method
if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orderList1 = [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)]
    print('Test1: Cost of', orderList1, 'is', buyLotsOfFruit(orderList1))
    orderList2 = [('apples', 2.0), ('pears', 3.0), ('banana', 4.0)]
    print('Test2: Cost of', orderList2, 'is', buyLotsOfFruit(orderList2))
```

C. shopSmart function

Instruction: Fill in the function `shopSmart(orders,shops)` in `shopSmart.py`, which takes an `orderList` (like the kind passed in to `FruitShop.getPriceOfOrder`) and a list of `FruitShop` and returns the FruitShop where your order costs the least amount in total. Don't change the file name or variable names, please. Note that we will provide the `shop.py` implementation as a "support" file, so you don't need to submit yours.

Method: The given class `FruitShop` has two instance variables `name` and `fruitPrices` and three instance method which are `getCostPerPound`, `getPriceOfOrder` and `getName`. We can use `FruitShop.getPriceOfOrder` to directly get the total cost of the input fruit list accordind to the fruitPrices of each shop.

I first consider that if the input list `fruitShops` is empty, there's no answer for the least total cost shop. So in this case, I'll return None.

Second if the input list `fruitShops` has only one shop, there is no doubt that this one is the answer, so I'll return the first element in the `fruitShops`.

Finally if the input list `fruitShops` has more than one shop, we should decide in which shop `orderList` costs the least amount. So I create a variable `bestShopIdx` to store the index of current best shop and then iterate over the `fruitShops` to compare the cost in current shop and in the current best shop. If the `orderList` cost less in current shop, it become the current best shop and `bestShopIdx` become the index of that shop. And after the iteration we have the index of best shop and just return `fruitShops[bestShopIdx]`.

Algorithm 2 shopSmart

Input: An list of (fruit,pound) tuples, orderList; An list of FruitShops, fruitShops;
Output: the best FruitShop where the orderList costs the least amount in total; or None;

if the length of $FruitShop \equiv 0$ **then**
    return $None$
else if the length of $FruitShop \equiv 1$ **then**
    return $FruitShop[0]$
else
    $bestShopIdx \leftarrow 0$
    for each $shop \in fruitShops$ with its index idx **do**
        if shop getPriceOfOrder($orderList$) < fruitShop[bestShopIdx] getPriceOfOrder($orderList$) **then**
            $bestShopIdx \leftarrow idx$
        end if
    end for
end if
return $fruitShops[bestShopIdx]$

Source Code

```
# shopSmart.py
"""
Here's the intended output of this script, once you fill it in:

Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
For orders: [('apples', 1.0), ('oranges', 3.0)] best shop is shop1
For orders: [('apples', 3.0)] best shop is shop2
"""

import shop


def shopSmart(orderList, fruitShops):
    """
        orderList: List of (fruit, numPound) tuples
        fruitShops: List of FruitShops
    """
    if len(fruitShops) == 0:
        return None
    elif len(fruitShops) == 1:
        return fruitShops[0]
    else:
        bestShopIdx = 0 # store the index of the current best shop
        for idx, shop in enumerate(fruitShops):
            if shop.getPriceOfOrder(orderList) < fruitShops[bestShopIdx].getPriceOfOrder(orderList):
                bestShopIdx = idx
        return fruitShops[bestShopIdx]


if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orders = [('apples', 1.0), ('oranges', 3.0)]
    dir1 = {'apples': 2.0, 'oranges': 1.0}
    shop1 = shop.FruitShop('shop1', dir1)
    dir2 = {'apples': 1.0, 'oranges': 5.0}
    shop2 = shop.FruitShop('shop2', dir2)
    shops = [shop1, shop2]
    print("For orders ", orders, ", the best shop is", shopSmart(orders, shops).getName())
    orders = [('apples', 3.0)]
    print("For orders: ", orders, ", the best shop is", shopSmart(orders, shops).getName())
```

III. DISCUSSION & CONCLUSION

In the first lab, I successfully build up my scientific Python environment. And I follow the tutorial using some of the Python packages to solve problems. It's a good opportunity to form a good programming base for future AI learning. And I will also be better at writing report in Latex by practicing in my each assignment. As practice makes perfect, let's work hard and make progress step by step.