U O W

Programming Basics for Big Data

CSCI316: Big Data Mining Techniques and Implementation

UNIVERSITY OF WOLLONGONG AUSTRALIA

Python for Data Analytics

- Why Python?
- Simple, and easy to read and learn
  - Python codes are often said as “executable pseudo-codes”
  - Much less verbose than Java
  - Allows you to focus on the algorithm of codes rather than being distracted by the syntax
- Powerful libraries for scientific computing, data analytics and machine learning
  - It is the most popular language in data science
- Drawbacks
  - May Not as fast as Java or C

Platforms and Libraries

Big Data Platform (Spark)

Deep Learning Platform (TensorFlow)

Data processing: Spark SQL
Dataframe, RDD,
Pandas on Spark
Scalable ML: MLlib

Data Analytics & ML Libraries in Python

Machine learning: Scikit-Learn
Data analytics: Pandas
Visualisation: Matplot-lib
Scientific computing: Numpy
Core: python core

UNIVERSITY OF WOLLONGONG AUSTRALIA

How you do implementation in this subject

- **Level-1**: Implement big data methods *from scratch*
  - For example, write your own code to implement an machine learning algorithm (instead of just calling Python's ML libraries directly)
- **Level-2**: Use specific Python's libraries (i.e., **Scikit-Learn**, **PySpark** and **TensorFlow**) to develop a big data project

A crash course on Python core and NumPy

All sample code is in the supplementary materials

Whitespace Format

- Many languages use curly braces to delimit blocks of code. Python uses **indentation**:

```
for i in [1, 2, 3, 4, 5]:
    print(i) # first line in "for i" block
    for j in [1, 2, 3, 4, 5]:
        print(i + j) # last line in "for j" block
print("done looping")
```

- Whitespace is ignored inside parentheses and brackets.

```
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 +
                           13 + 14 + 15 + 16 + 17)
easier_to_read_list_of_lists = [[1, 2, 3],
                                [4, 5, 6],
                                [7, 8, 9]]
```

Import Modules and Scripts

- Import libraries (or "modules" in Python's terms)

```
import math # import modules

math.ceil(1.1)  # equals 2
            # i.e. the least integer >= 1.1

ceil(1.1) #error: ceil is not defined

from math import ceil
ceil(1.1)   # equals 2

from math import * #import all functions in math
```


UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Functions

- Functions are defined using def:

```
def double(x):
    """this is where you put an optional docstring
    that explains what the function does.
    for example, this function multiplies its input by 2"""
    return x * 2


# call a function
y = double(3)
```

- Anonymous functions:

```
sum = lambda x, y: x + y
sum(3, 4)  # 7
```

Strings

- Strings can be delimited by single or double quotation marks

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
```

- Use backslashes to encode special characters

```
tab_string = "\t"
```

- Create raw strings using r"":

```
not_tab_string = r"\t"
        # represents the characters '\' and 't'
```

- Create multiline strings using triple-[double-]-quotes

```
multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```


UNIVERSITY OF WOLLONGONG AUSTRALIA

Lists

- The most fundamental data structure in Python

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [integer_list, heterogeneous_list, []]
list_length = len(integer_list)  # equals 3
list_sum = sum(integer_list)  # equals 6
```

```
x = [0, 1, ..., 9]
zero = x[0] # equals 0, lists are 0-indexed
one = x[1] # equals 1
nine = x[-1] # equals 9, for last element
eight = x[-2] # equals 8, for next-to-last element
x[0] = -1 # now x is [-1, 1, 2, 3, ..., 9]
```

```
first_three = x[:3] # [-1, 1, 2]
three_to_end = x[3:] # [3, 4, ..., 9]
one_to_four = x[1:5] # [1, 2, 3, 4]
```

- Lists are *mutable*

```
integer_list[1] = 0 # is [1, 0, 2]
```

Lists

- Concatenation

```
x = [1, 2, 3]
x.extend([4, 5, 6]) # x is now [1,2,3,4,5,6]
y = [1, 2, 3]
z = y + [4, 5, 6] # z is [1, 2, 3, 4, 5, 6]; y is unchanged
```

- Appendence

```
x = [1, 2, 3]
x.append(0) # x is now [1, 2, 3, 0]
y = x[-1] # equals 0
z = len(x) # equals 4
w = [x]
w.append([4, 5]) # w is now [[1, 2, 3, 0], [4, 5]]
```

- Unpack lists and the underscore sugar

```
x, y = [1, 2] # now x is 1, y is 2
_, y = [1, 2] # now y == 2, didn't care about the first element
```

Tuples

- Tuples are lists' immutable cousins.

```
my_tuple = (1, 2)
other_tuple = 3, 4
try:
    my_tuple[1] = 3
except TypeError:
    print("cannot modify a tuple")
```

- Represent multiple variables returned from functions

```
def sum_and_product(x, y):
    return (x + y), (x * y)
sp = sum_and_product(2, 3)  # equals (5, 6)
s, p = sum_and_product(5, 10)  # s is 15, p is 50
```

Dictionaries

- Dictionaries associate values with keys
  - Keys must be distinct
  - Allowing quick retrieval of a value corresponding to a given key

```
empty_dict = {}
grades = {"Joel": 80, "Tim": 95}  # dictionary literal
joels_grade = grades["Joel"]  # equals 80

"Joel" in grades  # True
not "Kate" in grades  # True

joels_grade = grades.get("Joel", 0)  # equals 80
kates_grade = grades.get("Kate", 0)  # equals 0
no_ones_grade = grades.get("No One")  # equals None

grades["Tim"] = 99  # replaces the old value
grades["Kate"] = 100  # adds a third entry
num_students = len(grades)  # equals 3
```

Dictionaries

```python
tweet = {
    "user": "joelgrus",
    "text": "Data Science is Awesome",
    "retweet_count": 100,
    "hashtags": ["#data", "#science",
                 "#datascience", "#awesome", "#yolo"]
}
tweet_keys = tweet.keys()  # list of keys
tweet_values = tweet.values()  # list of values
tweet_items = tweet.items()  # list of (key, value) tuples
"user" in tweet_keys  # True, but uses a slow list in
"user" in tweet  # more Pythonic, uses faster dict in
```

- Work with Counter

```python
from collections import Counter
c = Counter([0, 1, 2, 0])  # c is (basically) { 0 : 2, 1 : 1, 2 : 1 }
```

Sets

- A set represents a distinct list of elements

```
s = set()
s.add(1)  # s is now { 1 }
s.add(2)  # s is now { 1, 2 }
s.add(2)  # s is still { 1, 2 }
x = len(s)  # equals 2
y = 2 in s  # equals True
```

- Fast set membership check

```
stopwords_list = ["a", "an", "at"] + hundreds_of_other_words + ["yet", "you"]
"zip" in stopwords_list  # False, but have to check every element
stopwords_set = set(stopwords_list)
"zip" in stopwords_set  # very fast to check
```

Sets

- Find the distinct items in a collection

```
item_list = [1, 2, 3, 1, 2, 3]
num_items = len(item_list)  # 6
item_set = set(item_list)  # {1, 2, 3}
num_distinct_items = len(item_set)  # 3
distinct_item_list = list(item_set)  # [1, 2, 3]
```

- Set operations

```
t = set([1])
t.issubset(s)  # True
s.union(t)  # s
s.intersection(t)  # t
s.difference(t)  # {2}
```

Control Flow

```python
if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"

parity = "even" if x % 2 == 0 else "odd"

x = 0
while x < 10:
    print(x, "is less than 10")
    x += 1

for x in range(10):
    if x == 3:
        continue  # go immediately to the next iteration
    if x == 5:
        break  # quit the loop entirely
    print(x)  # returns 0,1,2,4
```

Sorting

- Sorting functions are built with Python lists

```
x = [4, 1, 2, 3]
y = sorted(x)  # is [1,2,3,4], x is unchanged
x.sort()  # now x is [1,2,3,4]
```

- Order and sorting parameter

```
# sort the list by absolute value from largest to smallest
x = sorted([-4, 1, -2], key=abs, reverse=True)  # is [-4,-2,1]

# sort the words and counts from highest count to lowest
word_counts = {"a": 1, "b": 3, "c": 2}
wc = sorted(word_counts.items(),
        key=lambda e: e[1],  # second element as key
        reverse=True)
# wc = [('b', 3), ('c', 2), ('a', 1)]
```

List Comprehensions

- Create a list, dictionary or set from a given list:

```
even_numbers = [x for x in range(5) if x % 2 == 0] # [0, 2, 4]
squares = [x * x for x in range(5)]  # [0, 1, 4, 9, 16]
even_squares = [x * x for x in even_numbers]  # [0, 4, 16]
square_dict = {x: x * x for x in range(5)}
                                        # { 0:0, 1:1, 2:4, 3:9, 4:16 }
square_set = {x * x for x in [1, -1]}  # { 1 }
zeroes = [0 for _ in even_numbers]
                                  # has the same length as even_numbers
pairs = [(x, y)
      for x in range(10)
      for y in range(10)]
                                  # 100 pairs (0,0) (0,1) ... (9,8), (9,9)
```


UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Function as Arguments: Map and Filter

- Map and Filter are useful operations especially in combination with in-line functions

```python
# map
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, items))
# Out: [1, 4, 9, 16, 25]

# filter
number_list = range(-5, 5)
less_than_zero = list(filter(
   lambda x: x < 0, number_list))
print(less_than_zero)
# Out: [-5, -4, -3, -2, -1]
```

Object-Oriented Programming

```
# by convention, we give classes PascalCase names
class Set:
    # these are the member functions
    # every one takes a first parameter "self" (another convention)
    # that refers to the particular Set object being used
    def __init__(self, values=None):
        # this is the constructor.
        self.dict = {} # instance property
        if values is not None:
            for value in values:
                self.add(value)

    # implement “add”
    def add(self, value):
        self.dict[value] = True

    # implement “contain”
    def contains(self, value):
        return value in self.dict
```

Object-Oriented Programming

- Use the implemented “Set” class:

```
s = Set([1, 2, 3])
print(s.contains(4)) # False
s.add(4)
print(s.contains(4)) # True
```

Numpy

- Numpy is the core library for scientific computing in Python.
- Basic Python math

```
a = [1, 2, -1, 4, 3]
max(a)  # 4
min(a)  # -1
sum(a)  # 9
```

- Numpy is the core library for scientific computing in Python.

```
np.mean(a)  # 1.5
np.var(a)  # variance, 2.96
np.median(a)    # 2.0
```

Numpy Array

- Numpy provides a high-performance multidimensional array object, and tools for working with these arrays.

```
import numpy as np
a = np.array([1, 2, 3])  # Create a rank 1 array
print(type(a))  # Prints "<class 'numpy.ndarray'>"
print(a.shape)  # Prints "(3,)"
print(a[0], a[1], a[2])  # Prints "1 2 3"
a[0] = 5  # Change an element of the array
print(a)  # Prints "[5, 2, 3]"
b = np.array([[1, 2, 3], [4, 5, 6]])  # Create a rank 2 array
print(b.shape)  # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])  # Prints "1 2 4"
```


UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Numpy

- Numpy data type
  - Every Numpy array is a grid of elements of the same type.
  - Numpy tries to guess a datatype, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

```
x = np.array([1, 2])  # Let numpy choose the datatype
print(x.dtype)        # Prints "int64"
x = np.array([1.0, 2.0])  # Let numpy choose the datatype
print(x.dtype)           # Prints "float64"d
x = np.array([1, 2], dtype=np.int64)  # Force a
                particular datatype
print(x.dtype)     # Prints "int64"
```


UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Numpy Array

- Array math (Matrix/Vector operations)

```
x = np.array([[1.0, 2.0], [3.0, 4.0]])
y = np.array([[5.0, 6.0], [7.0, 8.0]])

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

Numpy Array

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

Numpy Array

```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

v = np.array([9, 10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy Array

```
x = np.array([[1, 2],[3, 4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"

print(x)    # Prints "[[1 2]
            #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]
```